

## Advanced SAT-Techniques for Bounded Model Checking of Blackbox Designs \*

Marc Herbstritt    Bernd Becker    Christoph Scholl

Institute of Computer Science, Albert-Ludwigs-University  
79110 Freiburg im Breisgau, Germany

{herbstri, becker, scholl}@informatik.uni-freiburg.de

### Abstract

*In this paper we will present an optimized structural 01X-SAT-solver for bounded model checking of blackbox designs that exploits semantical knowledge regarding the node selection during SAT search. Experimental results show that exploiting the problem structure in this way speeds up the 01X-SAT-solver considerably. Additionally, we give a concise first-order formulation that is more expressive than using 01X-logic. This formulation leads to hard-to-solve QBF formulas for which experimental results from the QBF Evaluation 2006 are presented.*

### 1 Introduction

Today's verification of circuit designs heavily makes use of property checking. As algorithmic workhorse, model checking techniques are used. Within the large pool of different approaches, SAT-based bounded model checking [1] has become an established technique in academic research as well as in industrial tools. While typically the design under analysis is fully specified, the analysis of blackbox designs emerged as an interesting problem that has a large variety of applications [2, 3, 4], e.g., early design verification, error diagnosis, etc. A blackbox design corresponds to an incomplete circuit description, i.e., some parts of the circuits are not known.

In [5] we have proposed a SAT-based approach for bounded invariant checking of such blackbox designs. Since pure propositional logic is not expressive enough for the analysis of blackbox designs, one has to apply either logical extensions like 01X-logic or first-order logic where universal and existential quantifiers allow for a concise problem formulation. The usage of 01X-logic was analyzed

already in [5]. There, 01X-logic was algorithmically integrated into a structural SAT-solver. Then it was compared to an encoding approach in the style of [2]. As a result we have found that generally the encoding approach is much faster, although in some cases the structural 01X-SAT-solver performed better. The reason for that is the "blindness" of the encoding approach regarding the correspondence of And-Inverter-Graph (AIG) vertices that together build a semantical unit within 01X-logic.

In this work, we present a structural 01X-SAT-solver that combines the above mentioned encoding approach with a semantical node selection heuristics. Experimental results show that using this heuristics clearly outperforms the previous approaches both in CPU time used and in number of aborted instances.

Furthermore, we give for the first time a concise first-order formulation for the bounded model checking problem for blackbox designs with combinational blackboxes. To get correct results, the input-output-behavior of the blackboxes for equivalent input assignments within different time frames must be taken into account. This constraint leads to complex QBF formulas and we will present preliminary experimental data from the QBF Evaluation 2006 [6] showing that it requires sophisticated QBF-techniques to solve the corresponding QBF formulas.

The paper is structured as follows. In the following section, we review related work. In Section 3 we briefly give preliminaries necessary to understand the subsequent sections. Then, in Section 4 we describe our improved node selection heuristics when using 01X-logic to analyze blackbox problems. In this section we also present experimental results for our proposed optimization. Afterwards, in Section 5 we describe in detail how our QBF formulation works and present experimental results from the QBF solver evaluation in 2006. Finally, Section 6 concludes the paper.

\*This work was partly supported 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). See [www.avacs.org](http://www.avacs.org) for more information.

## 2 Related Work

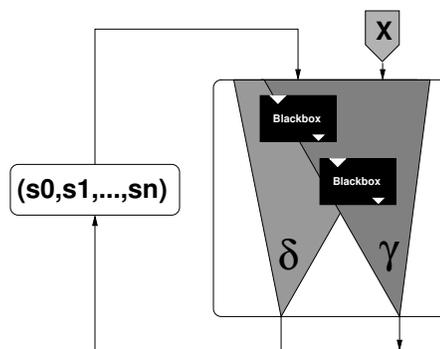
**Model Checking of Blackbox Designs** A first attempt for combinational equivalence checking of incomplete designs was made in [7] and further extended in [8]. In the context of symbolic CTL model checking, it turned out in [4] that modelling blackboxes by non-deterministic inputs ends up in ambiguous results when using different symbolic model checkers, e.g. VIS and SMV. Hence, in [4] approximation techniques were developed that give consistent results w.r.t. validity and realizability of incomplete designs. As already mentioned, in [5] a *bounded* model checking approach was presented that adapted the concept of Z-simulation, as it was used in [4] for BDD-based symbolic model checking, to the SAT-based bounded model checking framework. Recently, in [9] counterexample extraction within a BDD-based blackbox model checking framework was considered.

**01X-Logic** 01X-logic has one of its origins in ATPG where it was applied in circuit simulation to model the unpredictability of latches upon memory initialisation [10]. In [2] a transformation scheme was presented that compiles problems described in 01X-logic into pure propositional problems, allowing to use arbitrary engines that decide propositional logic.

**Quantified Boolean Formulas** Motivated by the impressive improvements for SAT engines within the last decade [11, 12, 13], current research tries to transfer this knowledge to the PSPACE-complete problem of deciding the validity of QBF [6]. Although current QBF solvers cannot compete generally with dedicated solvers for individual problems, the efficiency of modern QBF solvers has improved over the last years. It is also one aim of our work to trigger QBF-related issues and to support the QBF research community. The quest for the best algorithmic principle to decide QBF formulas is still open. Current QBF solvers are search-based in a DPLL-style (e.g., [14] and [15], expansion-based (e.g., [16]), or based on skolemization (e.g., [17]). Additionally, the usage of preprocessing techniques (see [18]) and the transformation into non-prenex form (see [19]) seem to be mandatory techniques to solve real world problems.

## 3 Preliminaries

We consider the analysis of sequential circuits where parts of the combinational logic are unknown, i.e., the circuit contains one or more blackboxes. In Figure 1 this scenario is visualized.



**Figure 1. Sequential circuit containing combinational blackboxes.**

### 3.1 Applications of Blackboxes

The concept of blackboxes can be used for various applications within a circuit design flow:

1. *Early Design Stage Verification.* Assume that two or more people are involved in designing a new circuit. Verification of properties or finding bugs, respectively, already in an early stage of the design, where at least one person has not finished yet its corresponding module, can be done by *blackboxing* the unimplemented module. Then, when a bug can be found independent of the blackbox, there must be an error outside the blackbox. Finding bugs in an early design stage can considerably save development costs.
2. *Module Abstraction.* Assume that a circuit developer wants to check some design property that does not depend on some modules of the design. These modules can be *blackboxed*, resulting in an abstraction of the circuit. This technique can be used, for example, when abstracting complex modules like multipliers or memory, decrease the required computational resources and thus enable the verification.
3. *Error Diagnosis.* Assume a bug was found and the circuit designer wants to know in which part of the design the error may be located. To do so, the designer can *blackbox* some candidate region in which the error is suspected. When blackbox analysis techniques can show that no error is observable, then it can be concluded that the candidate region contains the error location.

Modelling of the blackbox behaviour can be done in several ways. The most simple scheme relies on 01X-logic, a more advanced and more expressive scheme to be presented in Section 5 relies on Quantified Boolean Formulas.

AND <sub>01X</sub> (a,b)				OR <sub>01X</sub> (a,b)				NOT <sub>01X</sub> (a)	
a	b	0	1	X	a	b	0	1	X
0	0	0	0	0	0	0	0	1	X
1	0	0	1	X	1	1	1	1	1
X	0	0	X	X	X	X	X	1	X

**Table 1. Boolean operations AND, OR, and NOT extended to 01X-logic.**

### 3.2 01X-Logic

01X-logic extends propositional logic by a third logical value,  $X$ , to provide means to talk about the uncertainty of the status of propositional variables. Basic functions like conjunction, disjunction, and negation can be adapted conservatively by taking the controlling values of the gate function into account, as it is shown in Table 1.

The transformation scheme of [2] maps the three logical values 0, 1, and  $X$  to binary tuples  $(1,0)$ ,  $(0,1)$ , and  $(0,0)$ , respectively. Using this mapping, the extended operators can be adapted to this encoding:

$$\begin{aligned} \text{AND}_{01X}((g_0, g_1), (f_0, f_1)) &:= (g_0 + f_0, g_1 \cdot f_1) \\ \text{OR}_{01X}((g_0, g_1), (f_0, f_1)) &:= (g_0 \cdot f_0, g_1 + f_1) \\ \text{NOT}_{01X}((g_0, g_1)) &:= (g_1, g_0) \end{aligned}$$

### 3.3 Quantified Boolean Formulas

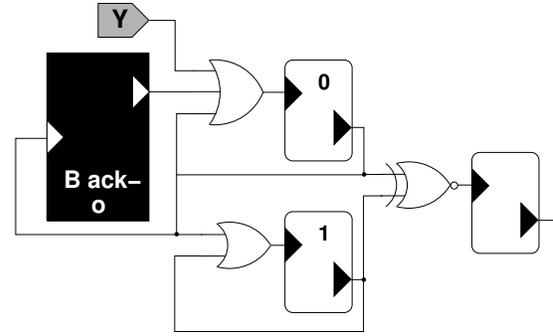
Quantified Boolean Formulas (QBFs) extend propositional formulas by allowing to *existentially* ( $\exists$ ) or *universally* ( $\forall$ ) quantify some variables. A QBF of the form  $\exists x : f(x)$  ( $\forall x : f(x)$ ) is called *valid* iff for some (all) value(s) of  $x \in \{0, 1\}$ ,  $f(x)$  evaluates to 1. Briefly, a QBF has the following general form:

$$\begin{aligned} Q_1 X_1 Q_2 Y_1 Q_3 X_2 Q_4 Y_2 \dots Q_n Y_{\frac{n}{2}} Q_{n+1} X_{\lceil \frac{n+1}{2} \rceil} : \\ f(X_1, Y_1, X_2, Y_2, \dots, Y_{\frac{n}{2}}, X_{\lceil \frac{n+1}{2} \rceil}), \end{aligned}$$

where  $X_1, Y_1, \dots, Y_{\frac{n}{2}}, X_{\lceil \frac{n+1}{2} \rceil}$  are pairwise disjoint sets of propositional variables,  $Q_1 = \exists, Q_i \in \{\exists, \forall\}, Q_i \neq Q_{i+1}, 1 \leq i \leq n$ , and  $f$  is a propositional formula in conjunctive normal form (CNF). Wlog. we require the first quantifier to be existential, but  $X_1$  may be empty, and the last quantifier to be existential, since innermost universally quantified variables can be immediately eliminated when  $f$  is given in conjunctive normal form. This form is typically used as input format by current QBF solvers.

### 3.4 And-Inverter-Graphs

The usage of And-Inverter-Graphs (AIGs) was mainly triggered by the work of Kuehlmann et al. [20] for space-efficient representation of boolean functions in the context



**Figure 2. Example showing feasibility of 01X-logic to detect counterexamples.**

of combinational equivalence checking and property checking. AIGs can be seen as combinational circuits restricted to two-input AND-gates which correspond to the vertices of the graph, and INV-gates which correspond to marked edges in the graph. The leaves of the graph are primary inputs. There exist efficient synthesis techniques based on structural and functional hashing to compute manageable AIGs for large circuits [20]. Figure 3 shows on the right two AIGs for the encoding of  $\text{AND}_{01X}$ .

## 4 Improvements to 01X-based BMC

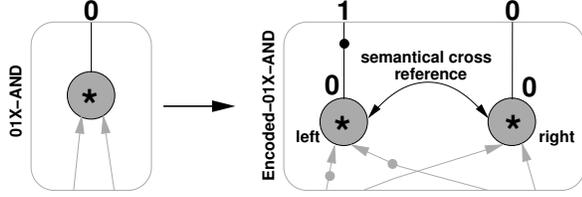
We start by looking at a small example showing that counterexamples can be found using 01X-logic. In Figure 2, a small sequential circuit with three flip-flops and one blackbox is shown. The next state functions, using 01X-logic, can be written as follows:

$$\begin{aligned} q'_0 &:= q_0 + Y + X \\ q'_1 &:= q_0 + q_1 \\ q'_2 &:= \neg(q_0 \oplus q_1) \end{aligned} \quad (1)$$

The property that should be checked is  $\text{AG}(q_0 \oplus q_1) = \text{AG}(\neg p)$ , i.e.,  $p$  should always be 0. We simulate the circuit under the semantics of 01X-logic, i.e., in each time step we assign the logical value  $X$  to the output of the blackbox. Starting in the initial state ( $q_0 = 0, q_1 = 1, p = 0$ ), the following table depicts a trace of length 3 that ends up in a state where  $p = 1$  holds, so that the property is violated:

step	y	q <sub>0</sub>	q <sub>1</sub>	p
0	—	0	1	0
1	1	1	1	0
2	0	1	1	1

In [5] the first author has implemented a SAT-based framework for bounded model checking using AIGs and a structural SAT-solver using AIGs as basic representation.



**Figure 3. Semantical cross reference between AIG vertices.**

The experiments in that work revealed that the encoding approach of Jain et al. [2] in general performs much more efficient than lifting the deduction rules of the structural SAT-solver to 01X-logic. But the experiments also showed that in some cases the SAT-solver can be misguided when using the encoding approach, because the variable selection mechanism of the propositional SAT-solver is not aware that the problem was generated from a 01X-logic problem. Put another way, the knowledge that two AIG vertices correspond to each other due to the binary encoding, was not available. Hence, one contribution of this work is to show that it pays off to take this correspondence into account during the search of the SAT-solver. To overcome the limitations of the previous approach, we propose an improved node selection heuristics that is used within our structural SAT-solver as follows.

The basic principle for our node selection heuristics is depicted in Figure 3. During construction of the AIG for the unfolding of the circuit under analysis, the two AIG vertices that are generated due the encoding approach are linked together by a semantical cross reference. The main drawback of the “blind” SAT-solver in [5] is that justifications of those two AIG vertices can be resolved at very different time points during the search. But now the semantical cross reference can be used to immediately handle AIG vertex justifications whenever the cross-referenced one was resolved. As an example have a look to the AIGs that refer to the encoding of an AND-gate, as depicted on the right hand side in Figure 3. I.e., assume that the justification of the left node was already resolved and recall that from the perspective of 01X-logic, we would like to justify the  $0_{01X} = (1, 0)$  value(s) of the encoding. Due to the semantical cross reference, we now can directly detect that the justification of the right AIG vertex should be handled. This strategy was integrated into the maximum-decision-level-heuristics already used in [5].

Table 2 shows experimental results for our new node selection heuristics.<sup>1</sup> As can be seen, it clearly outperforms the previous 01X-SAT-solver [5] not only in CPU time but also in the number of instances that can be solved.

<sup>1</sup>The CPU time limit was set to 900seconds. The times in column “Total” include the time for the aborted instances.

Solver	Time		#Solved / #Total
	Solved	Total	
“blind” SAT-solver [5]	964	17165	2712 / 2730
improved SAT-solver	386	12084	2717 / 2730

**Table 2. Results for the semantical node selection heuristics.**

## 5 QBF formulation

Before going into details how our QBF formulation works, we’ll have a look at a small example, showing that with 01X-logic, it is not always possible to detect a counterexample.

**Inaccuracy of 01X-Logic** For the circuit in Figure 4 the next state functions are as follows wrt. 01X-logic:

$$\begin{aligned}
 q'_0 &:= q_0 + X \\
 q'_1 &:= q_0 \cdot X \\
 q'_2 &:= 1 \\
 q'_3 &:= q_2 \\
 p' &:= (q_0 + (-q_1)) \cdot q_3 \cdot y
 \end{aligned} \tag{2}$$

The circuit is analyzed with respect to the property  $AG(y \cdot q_3 \cdot (-q_1 + q_0)) = AG(\neg p)$ , i.e.,  $p$  should always be 0. The following trace of length 4 shows that starting from the state  $(q_0 = 0, q_1 = 0, q_2 = 0, q_3 = 0, p = 0)$ , in the last step the signal value of  $(-q_1 + q_0)$ , that is necessary to be 1 so that  $p$  can be forced to 1, evaluates to  $X$ , and hence the property cannot be falsified.

step	$y$	$q_0$	$q_1$	$q_2$	$q_3$	$p$
0	—	0	0	0	0	0
1	—	X	0	1	0	0
2	—	X	X	1	1	0
3	1	X	X	1	1	X

As we will see, using QBF it is possible to detect a counterexample for the example given in Figure 4.

**Blackbox handling for QBF** Besides the issue of over-approximation as sketched in the example, the 01X-logic approach also abstracts from the fact that different blackbox outputs may compute different boolean functions. To take this into account, disjoint variables are introduced for the blackbox inputs and outputs. I.e., for the  $i$ th blackbox  $B_i$  that has  $k$  inputs ( $l$  outputs), we introduce variables  $(\xi_i^1, \xi_i^1, \dots, \xi_i^k)$  for the inputs, and variables  $(\gamma_i^1, \gamma_i^1, \dots, \gamma_i^l)$  for the outputs. We abbreviate these vectors using  $\Xi_i$  and  $\Gamma_i$ , respectively. Since we look at a finite unfolding, we additionally attach a *time index* to the variables, i.e., the  $k$ th

input of blackbox  $\mathcal{B}_i$  in time frame  $t$  corresponds to the variable  $\xi_{(i,t)}^k$ . The notion for the vectors is extended in the same manner, i.e.,  $\Gamma_{(i,t)}$  denotes the vector of variables that correspond to the outputs of blackbox  $\mathcal{B}_i$  in time frame  $t$ . In the following, we use  $in(\mathcal{B}_i)$  ( $out(\mathcal{B}_i)$ ) to denote the number of inputs (outputs) of blackbox  $\mathcal{B}_i$ .

**Transition Relation with Blackboxes** The above introduced variables for the blackbox outputs can now be taken into account when building the transition relation. I.e., the transition relation is built by symbolic simulation using AIGs as underlying data structure. During symbolic simulation, the blackbox outputs are handled as additional primary inputs. Regarding the implementation, we construct a *generic* transition relation  $T^{\text{gen}}(s, x, \Gamma_1, \dots, \Gamma_\beta, s')$  using generic versions  $\Gamma_i^{\text{gen}}$  of variables for the blackbox outputs of blackbox  $\mathcal{B}_i$ . For the construction of the finite unfolding of the transition relation, we use a substitution operator to replace all generic variables by their timed instantiation. In more detail, the transition relation  $T(s_i, x_i, \Gamma_{(1,i)}, \dots, \Gamma_{(\beta,i)}, s_{i+1})$ , describing the transitions from time frame  $i$  to time frame  $(i+1)$ , is obtained as follows:

$$T(s_i, x_i, \Gamma_{(1,i)}, \dots, \Gamma_{(\beta,i)}, s_{i+1}) := T^{\text{gen}}(s, x, \Gamma_1, \dots, \Gamma_\beta, s') \left\{ \begin{array}{l} s_i \leftarrow s \\ \Gamma_{(1,i)} \leftarrow \Gamma_1^{\text{gen}} \\ \Gamma_{(2,i)} \leftarrow \Gamma_2^{\text{gen}} \\ \dots \\ \Gamma_{(\beta,i)} \leftarrow \Gamma_\beta^{\text{gen}} \\ s_{(i+1)} \leftarrow s' \end{array} \right. \quad (3)$$

We apply the concept of *shadow*-variables suggested in [21] to efficiently implement this substitution. Now, the finite unfolding  $T(\beta, d)$  of the transition relation up to depth  $d$ , that takes  $\beta$ -many blackboxes into account, can be constructed:

$$T(\beta, d) := \bigwedge_{i=1 \dots (d-1)} T(s_i, x_i, \Gamma_{(1,i)}, \dots, \Gamma_{(\beta,i)}, s_{i+1}) \quad (4)$$

**Input-Output-Consistency** Since we focus on combinational blackboxes, we have to take care about the deterministic input-output-behaviour of the blackbox. Put another way, since the blackbox implements a function rather than a finite state machine, it has to produce the same output values given the same input values. Hence, we require a predicate to ensure this consistency constraint. For a blackbox  $\mathcal{B}_i$  and two different time frames  $t_1$  and  $t_2$ ,  $t_1 \neq t_2$ , the following predicate  $\text{IOC}(\mathcal{B}_i, t_1, t_2)$  is true iff the consistency constraint is not violated:

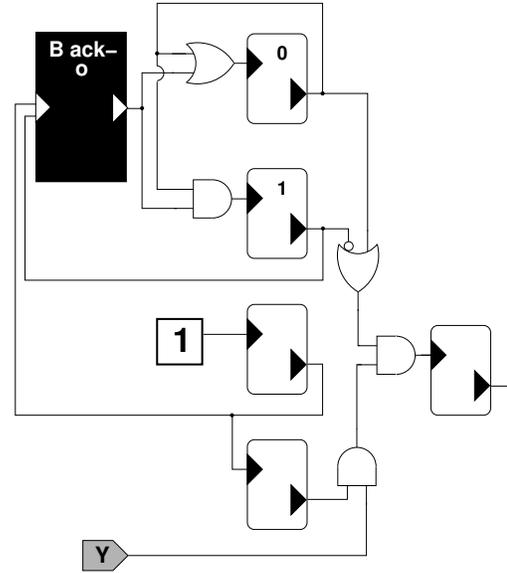


Figure 4. Example showing infeasibility of 01X-logic, but feasibility of QBF to detect counterexamples.

$$\text{IOC}(\mathcal{B}_i, t_1, t_2) := \left( \bigwedge_{m=1 \dots in(\mathcal{B}_i)} \xi_{(i,t_1)}^m \equiv \xi_{(i,t_2)}^m \right) \rightarrow \left( \bigwedge_{n=1 \dots out(\mathcal{B}_i)} \gamma_{(i,t_1)}^n \equiv \gamma_{(i,t_2)}^n \right) \quad (5)$$

This consistency constraint has to be fulfilled for all possible time frame combinations for a given maximum unfolding depth  $d$ . We capture this constraint using the predicate  $\text{IOC}(\mathcal{B}_i, d)$ :

$$\text{IOC}(\mathcal{B}_i, d) := \bigwedge_{\substack{1 \leq t_1, t_2 \leq d \\ t_1 \neq t_2}} \text{IOC}(\mathcal{B}_i, t_1, t_2) \quad (6)$$

Finally, the requirement for consistent input-output-behaviour must be satisfied for all blackboxes. Let  $\beta$  be the number of blackboxes in the original sequential circuit. Then,  $\text{IOC}(\beta, d)$  is true iff the consistency constraint for all blackboxes holds in the finite unfolding up to depth  $d$ :

$$\text{IOC}(\beta, d) := \bigwedge_{b=1 \dots \beta} \text{IOC}(\mathcal{B}_b, d) \quad (7)$$

**QBF Formula** Now we describe a QBF formulation for detecting counterexamples. Our QBF formula is *simulation driven*, since it follows the temporal behaviour of the sequential circuit and its finite unfolding, respectively. Put another way, we start in an initial state and select some assignment for the primary inputs. The values of the primary input and state variables induce deterministically values at

the blackbox inputs. The QBF-solver then has to check that *for all* possible value combinations at the blackbox outputs a next state is reached from which, again, a sequence of input assignments can be found such that a state violating the property is reached. For different instantiations of the same blackbox, but in different time frames, the input-output-consistency described above must be ensured, to avoid *false negative* counterexamples. Finally, we are able to construct a formula  $\Phi_{BMC}(d)$ :

$$\begin{aligned}
\Phi_{BMC}(d) := & \\
& \exists s^1 \exists x^1 \exists \Xi_{(1,1)} \forall \Gamma_{(1,1)} \dots \exists \Xi_{(\beta,1)} \forall \Gamma_{(\beta,1)} \\
& \exists s^2 \exists x^2 \exists \Xi_{(1,2)} \forall \Gamma_{(1,2)} \dots \exists \Xi_{(\beta,2)} \forall \Gamma_{(\beta,2)} \\
& \dots \\
& \exists s^{d-1} \exists x^{d-1} \exists \Xi_{(1,d-1)} \forall \Gamma_{(1,d-1)} \dots \exists \Xi_{(\beta,d-1)} \forall \Gamma_{(\beta,d-1)} \\
& \exists s^d : \\
& \text{IOC}(\beta, d) \rightarrow \left( I(s^1) \cdot T(\beta, d) \cdot (\neg P(s^d)) \right)
\end{aligned} \tag{8}$$

$I(s^1)$  is a predicate for the initial states, and  $P(s^d)$  describes the safety property in time frame  $d$ , i.e.,  $\neg P(s^d)$  states that the state reached after  $d$  time steps violates the given property. When formula  $\Phi_{BMC}(d)$  is *true*, then there exists a counterexample (in terms of a  $Q$ -model, see [15]) of length  $d$  such that for every blackbox implementation there exists an individual counterexample that corresponds to a path in the  $Q$ -model of  $\Phi_{BMC}(d)$ .

**Example Revisited** Let's see how the example from Figure 4 is handled with a QBF-solver using our encoding described above. Since within our QBF formulation, the blackbox outputs are handled by individual variables, the next state functions are rewritten, using the variable  $\gamma$  for the blackbox output.

$$\begin{aligned}
q'_0 & := q_0 + \gamma \\
q'_1 & := q_0 \cdot \gamma \\
q'_2 & := 1 \\
q'_3 & := q_2 \\
p' & := (q_0 + (\neg q_1)) \cdot q_3 \cdot \gamma
\end{aligned} \tag{9}$$

Figure 5 depicts a decision tree that is implicitly built by a QBF-solver when analyzing the QBF formula  $\Phi_{BMC}(3)$  for this example.

The left (right) edges correspond to assigning the blackbox output at some time frame to value 0 (1). The time frame can be derived from the depth of the source node of the edge. In the example, only in the last step the value of the primary input is of interest, hence in the diagram it is depicted on the bottom level only.

Solver	Time	#Solved/#Total
2clsQ	16828	0 / 28
GRL	16220	1 / 28
openQbf	16826	0 / 28
preQuantor	571	0 / 28
Qbfl	16792	0 / 28
Quaffle	16380	0 / 28
QUANTOR	906	0 / 28
QUANTOR_hc	900	0 / 28
qube3.0	16216	1 / 28
qube4.0	15828	1 / 28
qube5.0	20	28 / 28
semprop	16229	1 / 28
sKizzo-0.9-abs	9183	0 / 28
sKizzo-0.9-grn	2191	0 / 28
sKizzo-0.9.std	10761	0 / 28
SQBF	11359	0 / 28
sSolve	16808	0 / 28
ssolve+ut	16809	0 / 28
ssolve-ut	16809	0 / 28
WalkQSAT	16227	1 / 28
yQuaffle	16699	0 / 28

**Table 3. Short track results from QBFEVAL'06 for the `blackbox_design` family [6, 22].**

The decision tree can be read as follows: All path starting in the initial state lead to a state within 3 time steps whereby (1) the signal values along this path satisfy the input-output-consistency of Equation (7), and (2) the reached state violates the analyzed property, i.e.,  $p = 1$ . Hence, the sequence  $(-, -, 1)$  of values assigned to the primary input  $y$  is a counterexample. Please note that in general the counterexamples detected by our QBF formulation has not be uniform as it is the case for this small example.

**Results from QBF Evaluation 2006** Table 3 shows the short track results from the QBF Evaluation 2006 [6]. The benchmarks used are derived from our problem setting and are denoted `blackbox_design`. The benchmark set consists of 28 instances describing a blackbox bounded model checking problem for the `PicoJava/biu` benchmark from the `VIS` benchmark suite. The QBF instances are available from [22]. As can be seen from Table 3, only one out of 21 QBF-solvers, namely `qube5.0` (see [19] for current development of the QBF-solver `qube`), is able to solve all of the QBF instances. `qube5.0` applies transformations into a non-clausal representation and applies simplification and rewriting of the quantifier tree resulting in a non-prenex QBF. Additionally, `qube5.0` does efficient preprocessing of the original QBF, which drastically reduces the complexity of the QBF.

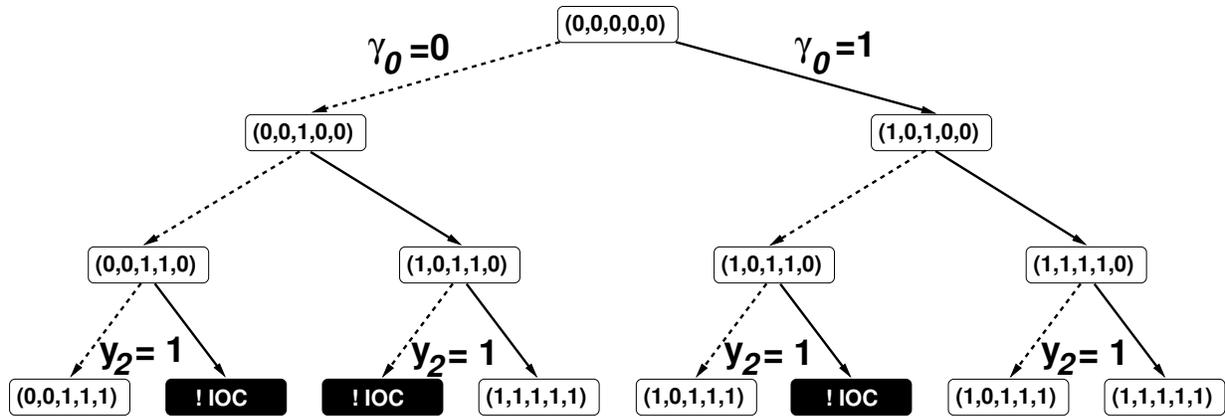


Figure 5. Decision tree implicitly built by QBF-solver. Tuples denote  $(q_0, q_1, q_2, q_3, p)$  and the depth of a node corresponds to the time frame within the unfolding.

## 6 Conclusions

In this paper we have reported on (1) optimizations for 01X-based bounded model checking of blackbox designs that considerably increase the efficiency of our 01X-SAT-solver, and (2) on a formalization of counterexamples using quantified boolean formulas that is more expressive than using 01X-logic. The QBF formulas turned out to be hard-to-solve for current state-of-the-art QBF solvers, but especially preprocessing techniques seem to make such a QBF-based approach viable.

As future work, we will investigate on how to combine 01X-logic and QBF. Furthermore, we will have a look on how the QBF formalization can be generalized to trade off the accuracy of the counterexamples and computational resources required to decide the existence of such a counterexample.

**Acknowledgements.** We are deeply grateful to Massimo Narizzano, Luca Pulina and Armando Tacchella for providing us the short track results of the QBF Evaluation 2006. Additionally, we would like to thank Tobias Nopper for contributing to the examples used in this paper.

## References

- [1] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.
- [2] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. Hsiao, "Testing, Verification, and Diagnosis in the Presence of Unknowns," in *Proc. of VLSI Test Symposium*, 2000, pp. 263–269.
- [3] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *Proc. of Design Automation Conference (DAC)*, 2001, pp. 238–243.
- [4] T. Nopper and C. Scholl, "Approximate symbolic model checking for incomplete designs," in *Proc. of 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, Nov 2004, pp. 290–305.
- [5] M. Herbstritt and B. Becker, "On SAT-based Bounded Invariant Checking of Blackbox Designs," in *Proc. of Microprocessor Test and Verification Workshop (MTV)*. Austin (TX), USA: IEEE Computer Society, 2005, pp. 23–28.
- [6] M. Narizzano, L. Pulina, and A. Tacchella, "QBF Evaluation 2006," available on-line at [www.qbflib.org/qbfeval](http://www.qbflib.org/qbfeval) [2006-08-02].
- [7] W. Günther, N. Drechsler, R. Drechsler, and B. Becker, "Verification of designs containing black boxes," in *EUROMICRO*, 2000, pp. 100–105.
- [8] C. Scholl and B. Becker, "Checking equivalence for circuits containing incompletely specified boxes." in *Proc. of 20th International Conference on Computer Design (ICCD)*, Freiburg im Breisgau, Germany, 2002, pp. 56–63.
- [9] T. Nopper and C. Scholl, "Counterexample generation for incomplete designs," in *ITG/GI/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, 2007, to appear.

- [10] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.
- [11] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. on Comp.*, vol. 48, no. 5, pp. 506–521, 1999.
- [12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proc. of Design Automation Conference (DAC)*, 2001.
- [13] N. Eén and N. Sörensson, "An extensible sat-solver." in *Proc. of 6th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2004, pp. 502–518, selected Revised Papers.
- [14] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Qube++: An efficient qbf solver." in *Proc. of 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, ser. Lecture Notes in Computer Science, vol. 3312. Austin, Texas, USA: Springer, 2004, pp. 201–213.
- [15] H. Samulowitz and F. Bacchus, "Using sat in qbf." in *Proc. of 11th International Conference on Principles and Practice of Constraint Programming (CP)*, ser. Lecture Notes in Computer Science, vol. 3709. Sitges, Spain: Springer, 2005, pp. 578–592.
- [16] A. Biere, "Resolve and expand." in *Proc. of 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, ser. Lecture Notes in Computer Science, vol. 3542. Vancouver, BC, Canada: Springer, 2005, pp. 59–70, selected Papers.
- [17] M. Benedetti, "skizzo: A suite to evaluate and certify qbfs." in *Proc. of 20th International Conference on Automated Deduction (CADE)*, ser. Lecture Notes in Computer Science, vol. 3632. Tallinn, Estonia: Springer, 2005, pp. 369–376.
- [18] H. Samulowitz, J. Davies, and F. Bacchus, "Preprocessing qbf." in *Proc. of 12th International Conference on Principles and Practice of Constraint Programming (CP)*, ser. Lecture Notes in Computer Science, vol. 4204. Springer, 2006, pp. 514–529.
- [19] E. Giunchiglia, M. Narizzano, and A. Tacchella, "Quantifier structure in search based procedures for qbfs." in *Proc. of Conference on Design, Automation and Test in Europe (DATE)*. Munich, Germany: European Design and Automation Association, 2006, pp. 812–817.
- [20] A. Kuehlmann, V. Paruthi, F. Krohm, and M. M.K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *IEEE Trans. on CAD*, 2002.
- [21] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking." in *Proc. of International Conference on Computer-Aided Design (ICCAD)*. San Jose, CA, USA: IEEE Computer Society / ACM, 2004, pp. 50–57.
- [22] M. Herbstritt, "QBF blackbox\_design family benchmarks," available on-line at [http://www.qbflib.org/suite\\_detail.php?suiteId=22](http://www.qbflib.org/suite_detail.php?suiteId=22) [2006-08-02].