Propositional Approximations for Bounded Model Checking of Partial Circuit Designs*

Bernd Becker, Marc Herbstritt, Natalia Kalinnik, Matthew Lewis, Juri Lichtner, Tobias Nopper, Ralf Wimmer
Albert-Ludwigs-University, Freiburg im Breisgau, Germany
{becker|herbstritt|kalinnik|lewis|lichtner|nopper|wimmer}@informatik.uni-freiburg.de

Abstract—Bounded model checking of partial circuit designs enables the detection of errors even when the implementation of the design is not finished. The behavior of the missing parts can be modeled by a conservative extension of propositional logic, called 01X-logic. Then the transitions of the underlying (incomplete) sequential circuit under verification have to be represented adequately. In this work, we investigate the difference between a relation-oriented and a function-oriented approach for this issue. Experimental results on a large set of examples show that the function-oriented representation is most often superior w.r.t. (1) CPU runtime and (2) accuracy regarding the ability to find a counterexample, such that by using the function-oriented approach an increase of accuracy up to 210% and a speed-up of the CPU runtime up to 390% compared to the relation-oriented approach are achieved. But there are also relevant examples, e.g. a VLIW-ALU, for which the relation-oriented approach outperforms the function-oriented one by 300% in terms of CPU-time, showing that both approaches are efficient for different scenarios.

I. INTRODUCTION

Within the last ten years, bounded model checking (BMC) has emerged as an effective verification technique for finding counterexamples of erroneous circuit designs [1], [2]. BMC makes use of a SAT-solver for the satisfiability problem, and due to the enormous performance increase of SAT-solvers within the last years, SAT-based BMC is applicable to large industrial designs. While the circuit design under verification is typically considered to be complete, in this work we focus on partial designs for which some parts of the circuit design are not implemented yet. To enable the analysis of such partial designs with BMC, the missing parts have to be modeled adequately, e.g. by using 01X-logic, a conservative extension of propositional logic that introduces a third logical value \( X \) by using 01X-logic, a conservative extension of propositional logic to express that the transition function. The \( \text{compose} \)-operation that underlies the functional approach is performed directly on the AIG. Together with the initial condition and the property to check, a CNF-formula is generated that is checked by a SAT-solver.

Transition systems form the basis of sequential circuits. In this work we analyze two different ways for the representation of transitions of partial sequential circuit designs that are analyzed via 01X-BMC: (1) A relation-oriented approach and (2) a function-oriented approach. For complete circuits both approaches are equal to each other. But for partial circuit designs it turns out that the relation-oriented approach is less accurate than the function-oriented approach. Furthermore, the function-oriented approach often results in a SAT-instance that is simpler to solve than the corresponding SAT-instance of the relation-oriented approach. But our experimental results also show that there are relevant cases, e.g. an erroneous VLIW-ALU, for which the relation-oriented approach is faster than the function-oriented approach and accurate enough to find a counterexample.

As a summary, both the relation-oriented approach and the function-oriented approach together build an efficient tool-box for bounded model checking of partial designs.

The paper is structured as follows. After presenting related work in the next section, we present preliminaries in Sect. II. I. In Sect. IV we discuss the relationship between a relation-oriented and function-oriented approach for transition representation in the context of 01X-BMC. Experimental results are presented and analyzed in Sect. V. The paper concludes with Sect. VI.

II. RELATED WORK

SAT-based BMC of partial circuit designs was investigated in a relational style in [3], [4], [5] where BMC was adapted to 01X-logic as well as to Quantified Boolean Formulas (QBFs). However, these works show the feasibility of 01X-BMC for a large number of relevant examples and hence served as starting point for our investigations.

In the area of BDD-based symbolic model checking [6], [7], both relational and functional approaches are common methods for pre-image computation. Functional pre-image computation has the advantage of a lower need for symbolic variables that usually leads to a problem instance that is simpler to solve, but the overall time for verification very much depends on the circuit under verification and hence relational pre-image computation may also be effective [8], [9].

For complete designs, [10] presents a BMC technique that uses And/Inverter-graphs (AIGs) [11], [12] to represent the transition function. The \( \text{compose} \)-operation that underlies the functional approach is performed directly on the AIG. Together with the initial condition and the property to check, a CNF-formula is generated that is checked by a SAT-solver.

Symbolic trajectory evaluation (STE) [13], [14], [15] is a related method that is also based on ternary 01X-logic. However, based on the formula under test, STE performs an automated 01X-abstraction of an initially \emph{complete} circuit design, and hence it is not directly applicable to partial circuit designs. We assume that STE may also be adapted to partial circuit designs, but we do not consider this case in the paper at hand.

* 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 for more information.
In [16], a BDD-based approach is presented to symbolically compute counterexamples for erroneous partial circuit designs. Hence, the work of [16] comes closest to the problem setting discussed in this paper. While the method of [16] is based on BDDs (see also [17], [18]), our methods rely on SAT-solvers. We expect that a BDD-based and a SAT-based approach complement each other, which has to be shown in future work.

III. PRELIMINARIES

A. Bounded Model Checking

Model checking [19] is a verification technique to certify the correctness of sequential systems w.r.t. a temporal property that is used as a specification. In [1], [2] Biere et al. introduced a variant called bounded model checking (BMC) that focuses on the falsification of properties, i.e., on the generation of a counterexample in case that the design is erroneous.

For the work at hand, we consider sequential circuits \((x, s, \delta, \lambda)\) with \(n\) primary inputs \(x = (x_0, \ldots, x_{n-1})\), a register \(s = (s_0, \ldots, s_{p-1})\) of \(p\) latches to store intermediate values, the next-state functions \(\delta = (\delta_0, \ldots, \delta_p)\) that compute the next-state values of the latches, and the output functions \(\lambda = (\lambda_0, \ldots, \lambda_q)\) for computing the values at the primary outputs. The latches impose a discrete state space onto the system that consists of all possible combinations of latch values. Each such combination is called a state. Each function \(\delta_i, \lambda_j\) is a boolean function \(B^p \times B^q \rightarrow B\).

Let \(x'(s')\) be the primary inputs (state variables) at time step \(i\), respectively. Then, the value of latch \(l\) in the next time step is computed by \(s'_{i+1} = \delta_l(x', s')\). With this we can define a transition relation \(T(s', x', s_{i+1}) = \bigwedge_{i=0}^{p-1} (s'_{i+1} \equiv \delta_l(s', x'))\) which is true if there is a transition from state \(s'\) to state \(s_{i+1}\) by applying input \(x'\).

An invariant \(p\) is a property that should always hold and is described by using the state variables \(s^k\), i.e., we have a predicate \(P(s^k)\) that describes that \(p\) holds in state \(s^k\). Finally, the BMC-formula for checking whether a state can be reached within \(k\) time steps such that property \(p\) is violated is as follows:

\[
\text{BMC}(k) = I(P(0)) \land T(s^0, x^0, s^1) \land \cdots \land T(s^{k-1}, x^{k-1}, s^k) \land \neg P(s^k).
\]

This formula can be translated into conjunctive normal form (CNF), such that an off-the-shelf SAT-solver can be applied. If the SAT-solver finds a satisfying assignment for \(\text{BMC}(k)\), then this satisfying assignment corresponds to a counterexample that testifies the violation of property \(p\).

B. 01X-based Bounded Model Checking

We consider sequential circuit designs where parts of the combinational logic are unknown. The missing parts are denoted as blackboxes. A blackbox is a combinational module of which we do not know the boolean function that it computes. In Figure 1 this scenario is visualized. The question we want to answer by BMC is: Is a given invariant property violated independently of the implementation of the blackboxes?

Blackboxes play an important role in the design flow of digital systems: (1) They can be used for verification in an early stage of the design process when not all modules are implemented yet. (2) The verification of complex systems can be simplified by putting parts of the design, on which the property under consideration may not depend, into blackboxes, and (3) blackboxes can be used for fault localization: If a fault was found that disappears when a certain region of the circuit design is put into a blackbox, this “blackboxed” region is a good candidate where the fault may be located.

For BMC, the unknown behavior of blackboxes can be modeled using a three-valued logic. Adding a third logical value to propositional logic that represents the uncertainty about values of propositional variables results in 01X-logic. The basic boolean operators, AND, OR, and NOT, can be extended to 01X-logic in a conservative way, see Table I.

In [20], Jain et al. have proposed an approach for handling the three logical values 0, 1, and \(X\) by applying a binary encoding and extending this encoding to the operators as follows. The logical values 0, 1, and \(X\) are binary encoded as \(0_{\text{aux}} := (1, 0), 1_{\text{aux}} := (0, 1), \) and \(X_{\text{aux}} := (0, 0), \) resp., and the basic operators are adapted, now using tuples \((a_0, a_1)\) and \((b_0, b_1)\) for 01X-variables, as \(\text{AND}_{\text{aux}} ((a_0, a_1), (b_0, b_1)) := (a_0 \cdot b_0, a_1 \cdot b_1), \text{OR}_{\text{aux}} ((a_0, a_1), (b_0, b_1)) := (a_0 \cdot b_0, a_1 + b_1), \) and \(\text{NOT}_{\text{aux}} ((a_0, a_1)) := (a_1, a_0)\).

C. Relational and Functional Transition Representation

In Formula (1) a relational representation of the transitions was used, but we may also use a functional representation. We now describe both approaches in more detail.

For the relational approach, we make use of the \(k\)-step transition relation \(T^k\) that is given by

\[
T^k(s^0, x^0, \ldots, x^{k-1}, s^k) := \bigwedge_{i=1}^{k} T(s^{i-1}, x^{i-1}, s^i)
\]

\[
= \bigwedge_{i=1}^{k} \left( s'_{i} \equiv \delta(s^{i-1}, x^{i-1}) \right) = \bigwedge_{i=1}^{k} \bigwedge_{j=0}^{p-1} \left( s'_{i} \equiv \delta_j(s^{i-1}, x^{i-1}) \right)
\]

For the functional approach, instead, we make use of the \(k\)-step transition function \(\delta^k : B^p \times (B^q)^k \rightarrow B\) that is inductively defined by

\[
\delta^k = \delta^{k-1} \circ \delta^1 = \delta^k \circ \delta^0
\]
defined for a latch \( l \) as follows:

\[
\delta_l^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}) = \delta_l^k(\mathbf{s}^{k-1}(\mathbf{0}, x^0, \ldots, x^{k-2}), \ldots, \delta_l^0(\mathbf{s}^{k-1}(\mathbf{0}, x^0, \ldots, x^{k-2})), \mathbf{x}^k)
\]

i.e., the input \( \delta_{p-1}^k \) is substituted by the \((k-1)\)-step transition function of latch \( l^p \). Finally, we collect all local \( k \)-step transition functions within a vector \( \mathbf{\delta}^k \):

\[
\mathbf{\delta}^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}) := (\delta_l^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}), \ldots, \delta_{p-1}^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}))
\]

Assume that for BMC of a complete circuit design the set of initial states is given by a predicate \( I(\mathbf{s}^0) \) and for a specified unfolding depth \( k \) the invariant to be checked is given by \( P(\mathbf{x}^k) \). Then, the formulas

\[
\begin{align*}
\text{BMC}^c(k) &:= I(\mathbf{s}^0) \cdot \mathbf{T}^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}, \mathbf{s}^k) \cdot P(\mathbf{x}^k) \\
\text{BMC}^f(k) &:= I(\mathbf{s}^0) \cdot \mathbf{P}(\mathbf{\delta}^k(\mathbf{s}^0, x^0, \ldots, x^{k-1}))
\end{align*}
\]

are satisfiable iff there exists a counterexample of length \( k \) that falsifies property \( p \). The formula \( \text{BMC}^c \) (\( \text{BMC}^f \)) is called the relational (functional) BMC formula, resp.

We demonstrate the individual procedure regarding both the functional and the relational transition representation within BMC using \( \text{BMC}^c \) and \( \text{BMC}^f \), resp., by a small example. This is done for a sequential circuit without blackboxes. In the subsequent Sect.IV-B we are making the case for a sample design containing blackboxes.

Let us consider the circuit in Figure 2(a). We describe step by step the proceedings of BMC towards a CNF. The property that is checked is given by \( \mathcal{A}(\mathbf{s}_0, \mathbf{s}_1, \ldots) \), i.e., none of both latches should ever be 1. The initial state of our system is given by \( \mathbf{s}_0 = 0 \) and \( \mathbf{s}_1 = 0 \) and hence the initial state predicate is \( I(\mathbf{s}^0) = \mathbf{s}_0 \cdot \mathbf{s}_1 \).

The state transition functions at time step \( i \) are given as

\[
\delta_l(\mathbf{s}', \mathbf{s}) = \mathbf{s}_0 + \mathbf{s}_1 + \mathbf{s}'
\]

and

\[
\delta_l(\mathbf{s}', \mathbf{s}) = 1.
\]

Furthermore, the property to verify at time step \( i \) is constituted by \( P(\mathbf{x}') = \mathbf{s}_0 \cdot \mathbf{s}_1 \) or equivalently \( P(\mathbf{x}') = \mathbf{s}_0 + \mathbf{s}_1 \). Please note that this is a very simple example, since after the first transition step the property is violated immediately; it should mainly serve as a traceable example showing the difference between the relational and the functional approach for BMC.

First we start with the relational approach and recall Equation (2). In our example this leads to the following formula:

\[
T(\mathbf{s}', \mathbf{s}) = (\mathbf{s}'' \equiv \delta_l(\mathbf{s}', \mathbf{s}')) \cdot (\mathbf{s}' \equiv \delta_l(\mathbf{s}', \mathbf{s}')) \cdot (\mathbf{s} \equiv 1)
\]

by using the equivalence-operator \( \equiv \), which itself can be expressed by negating the boolean difference. Hence, (10) becomes \( \mathbf{s}_0 \equiv \delta_l(\mathbf{s}, \mathbf{x}) \), which can be written as

\[
\mathbf{s}_0 \equiv \delta_l(\mathbf{s}, \mathbf{x}) + \mathbf{s}_0 \cdot \overline{\delta_l(\mathbf{s}, \mathbf{x})}\n\]

using only AND, OR, and NOT. Now we look at the binary encoding of expression (11). We use \( k \) instead of \( s_i^l \) and \( l \) instead of \( \delta_l(\mathbf{s}, \mathbf{x}) \), resp., to shorten writing. Assume \((k_0, k_1)\) and \((l_0, l_1)\) to be the tuples used for the binary encoding of \( k \) and \( l \). Then,
ENC−AND
ENC−AND
0 1
1
1 1 1
0 0 0

That means, the global transition relation for both operands are equal. This is in fact true, since \( k \) may be 1 and \( l \) may be 01X. Substituting these values into Equation (12) results in computing a value \( (0, 0) =: X_{01X} \). Is this correct? We expect from the equivalence operator that it results in \( 0 \) when both operands are equal, and \( 0 \) otherwise. But our 01X-abstraction leads to the inconclusive result \( X_{01X} \), meaning that we do not know whether both operands are equal. This is in fact true, since \( k \) may still be 1 and \( l \) may be 0 and thus would not be equal. Nevertheless, this observation will pose a problem that will be discussed in the following.

How is this issue related to our relational representation of the transitions? The transition relation was built by equating next-state-variables with their next-state-functions, see Equation (10). That means, the global transition relation \( T(s, x, x') \) is computed by a conjunction of all local transition relations \( T_i(s, x, x'_i) \), i.e.,

\[
T(s, x, x') := \bigwedge_{l=0}^{p-1} T_l(s, x, x'_l) = \bigwedge_{l=0}^{p-1} (s'_l \equiv \delta_l(s, x)).
\]

Recall that our BMC-formula consists of three parts: \( \text{BMC}(k) = I(s^0) \cdot \left( \bigwedge_{l=0}^{k-1} T(s^l, x^l, x'^{l+1}) \right) \cdot P(s^k) \), namely the predicate for the initial states, the predicate for the \( k \)-fold unfolding of the transition relation, and the predicate for the specification property. The SAT-problem corresponding to \( \text{BMC}(k) \) is written more precisely as

\[
\exists s^0 \exists x^0 \exists x^1 \exists x^2 \ldots \exists x^{k-1} \exists x^k : \quad I(s^0) \cdot \left( \bigwedge_{l=0}^{k-1} T(s^l, x^l, x'^{l+1}) \right) \cdot P(s^k) = 1.
\]

Hence, a satisfying solution \( \varphi_{\text{CE}} \) for formula (14) requires that each predicate \( I(s^0), T(s^l, x^l, x'^{l+1}) \), and \( P(s^k) \) is satisfied for \( 0 \leq i < k \). Especially for the transition relation this means that in each unfolding step \( i \) all local transition relations have to be satisfied, i.e., \( T_i(s^l, x^l, x'^{l+1}) = 1 \). Regarding our binary encoding for 01X-logic, \( \text{BMC}(k) \) is encoded into a tuple \( (\text{BMC}(k)_0, \text{BMC}(k)_1) \). Since \( 1_{01X} \) is encoded as \( (0, 1) \), our propositional satisfiability problem for the 01X-based approach, i.e., \( (\text{BMC}(k)_0, \text{BMC}(k)_1) = (0, 1) \), results in solving the following propositional problem \( \text{BMC}(k)_0 \cdot \text{BMC}(k)_1 = 1 \). The satisfiability requirement directly forces implications on the sub-formulas for \( \text{BMC}(k) \). This scenario is depicted in Figure 2(b). The 0-values are implicitly implied, since for encoding tuples \( (e_0, e_1) \) only one of both variables \( e_0 \) or \( e_1 \) can be assigned to 1, because \( (1, 1) \) does not correspond to a valid 01X-value. Please note that especially for the encoding tuple \( (T^k(s^0, x^0, \ldots, x^{k-1}, s^k)_0, T^k(s^0, x^0, \ldots, x^{k-1}, s^k)_1) \) of the \( k \)-fold unfolding of the transition relation, the value \( (0, 1) =: 1_{01X} \) is opposed.

Consequently, for each global transition relation \( T(s^l, x^l, x'^{l+1}) \) in unfolding step \( i \), this implies for its binary encoding that \( (T^l_0, T^l_1) = (0, 1) \) must hold. And by applying this implication argument one more time, we end up that for each local transition relation \( T_i(s^l, x^l, x'^{l+1}) \) the following condition must also hold:

\[
(T_i(s^l, x^l, x'^{l+1})_0, T_i(s^l, x^l, x'^{l+1})_1) = (0, 1).
\]

Now, by substituting \( k := T_i(s^l, x^l, x'^{l+1})_0 \) and \( l := T_i(s^l, x^l, x'^{l+1})_1 \) in Equation (12) for computing \( T_i(s^l, x^l, x'^{l+1})_0 \), \( T_i(s^l, x^l, x'^{l+1})_1 \) results in computing

\[
(k_0 \cdot l_0 + k_1 \cdot l_1 + k_2 \cdot l_0 + k_1 \cdot l_1) \cdot (k_0 \cdot l_0 + k_1 \cdot l_1).
\]

which can be simplified to

\[
k_0 \cdot k_1 \cdot l_1 \cdot l_0 + k_0 \cdot k_1 \cdot l_0 \cdot l_1.
\]

The function of (17) computes a 1 iff \( (k_0, k_1) = (0, 1) = 1_{01X} \) and \( (l_0, l_1) = (0, 1) = 1_{01X} \), or \( (k_0, k_1) = (1, 0) = 0_{01X} \) and \( (l_0, l_1) = (1, 0) = 0_{01X} \). Hence, for the propositional fragment of 01X-logic, we get correct results regarding equivalence, but for values \( X_{01X} \) our equivalence operator gives negative results.
But what exactly is the problem then? If we would indeed like to check equivalence of two 01X-values, the above described approach is absolutely correct: In case that \( \delta = X_{\text{orr}} \) and \( l = X_{\text{orr}} \) we cannot guarantee that, e.g., \( \delta \) may have the value 0 and \( l \) may have the value 1. Thus, our 01X-operator for equivalence gives the correct answer by computing \( X_{\text{orr}} \). But for our relational approach, we have abused the equivalence-operator for storing the output of the transition function \( \delta_i(s,x) \) in the variable \( s'_f \) for the next state. The difference is that we do not require equivalence in the logical sense, but equivalence in a semantical sense, i.e., \( s'_f = \delta_i(s,x) \) should be true also when \( s_f = X_{\text{orr}} \) and \( \delta_i(s,x) = X_{\text{orr}} \).

This artefact leads to a coarser approximation of the state transitions that may be selected during the SAT-solving. Hence, the relation-oriented approach is less accurate than the function-oriented one, as illustrated in the following.

**B. Example**

We demonstrate the different behavior in terms of accuracy of the relation-oriented and the function-oriented approach for a small example. The partial circuit design of Figure 2(c) consists of two latches, both initialized to value 0. We assign the value \( X_{\text{orr}} \) to the output of the blackbox. Again, the property \( \forall \exists T(\neg \delta \cdot \neg \tau) \) is checked, resulting in \( P(s'_f) = s_f \cdot s_i \).

Then we obtain the following one-step transition functions: \( \delta_i(s_0,s_1,x) = s_1 + X_{\text{orr}} \) and \( \delta_i(s_0,s_1,x) = 1 \). Doing so, we get the following relational 01X-BMC formula:

\[
\begin{align*}
\text{BMC}^{01X}(1) &= I(s_0, s'_1) \cdot \left( \text{and} \left( s'_0, s'_1, x, s_0 \right) \right) \cdot P(s'_0, s'_1) \\
&= \left( \overline{s'_0 + s'_1} \right) \cdot \left( s'_0 + x_{\text{orr}} \right) \cdot \left( s'_1 + 1 \right) \cdot \left( s'_0 + s'_1 \right) \\
&= \left( \overline{s'_0 + s'_1} \right) \cdot \left( s'_0 + x_{\text{orr}} + s'_0 \cdot \overline{s'_1 + x_{\text{orr}}} \right) \cdot \left( s'_1 + s'_1 \right) \\
&= \left( \overline{s'_0 + s'_1} \right) \cdot \left( s'_0 + x_{\text{orr}} + s'_0 \cdot \overline{s'_1 + x_{\text{orr}}} \right) \cdot \left( s'_1 \cdot \left( s'_0 + s'_1 \right) \right).
\end{align*}
\]

We set \( s'_f := (s'_0, s'_1, 1) \) and apply the Jains-encoding from Section III-B to obtain the following boolean formula:

\[
\text{BMC}^{\text{enc}}_{01X}(1) = \text{AND}_{\text{orr}} \left( \begin{array}{c}
\text{NOT}_{\text{orr}}(s'_0, s'_1, 0), \\
\text{OR}_{\text{orr}} \left( \text{AND}_{\text{orr}} \left( s'_0, s'_1, 0, 1 \right), \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 0 \right) \right), \\
\text{AND}_{\text{orr}} \left( \text{NOT}_{\text{orr}} \left( s'_0, s'_1, 0 \right) \right), \\
\text{NOT}_{\text{orr}} \left( \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 1 \right), \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 0 \right) \right) \right) \cdot \left( s'_1 \cdot s'_1 \right) \cdot \left( s'_0 + s'_1 \right).
\end{array} \right)
\]

For Formula (19) to evaluate to \( 1_{\text{orr}} = (0, 1) \), we need to check the satisfiability of the following formula:

\[
\left( s'_0 + s'_1 + s'_0 \cdot 1 \cdot s'_0 + s'_0 \cdot 0 + s'_0 \cdot 1 \cdot s'_1 \cdot 1.1 \right)
\]

Applying de Morgan’s rule we obtain

\[
\left( s'_0 + s'_1 + s'_0 \cdot 1 \cdot s'_0 + s'_0 \cdot 0 + s'_0 \cdot 1 \cdot s'_1 \cdot 1.1 \right)
\]

It is easy to see that Formula (21) is unsatisfiable, since it contains both \( s'_f \) and \( s'_1 \) as unit clauses. Hence, no counterexample can be found by the relation-oriented approach.

For the function-oriented approach, we get the following 01X-BMC formula:

\[
\text{BMC}_{01X}^{f}(1) = \left( \overline{s'_0 + s'_1} \right) \cdot \left( s'_0 + \overline{s'_1 + x_{\text{orr}}} \right) + 1.
\]

We again set \( s'_f := (s'_0, s'_1, 1) \) and apply the Jain-encoding from section III-B to obtain the following formula:

\[
\text{BMC}^{\text{enc}}_{01X}(1) = \text{AND}_{\text{orr}} \left( \begin{array}{c}
\text{NOT}_{\text{orr}}(s'_0, s'_1, 0), \\
\text{OR}_{\text{orr}} \left( \text{AND}_{\text{orr}} \left( s'_0, s'_1, 0, 1 \right), \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 0 \right) \right), \\
\text{AND}_{\text{orr}} \left( \text{NOT}_{\text{orr}} \left( s'_0, s'_1, 0 \right) \right), \\
\text{NOT}_{\text{orr}} \left( \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 1 \right), \text{OR}_{\text{orr}} \left( s'_0, s'_1, 0, 0 \right) \right) \right) \cdot \left( s'_1 \cdot s'_1 \right) + \left( s'_0 + s'_1 \right).\]

For the resulting tuple of Formula (23) to evaluate to \( 1_{\text{orr}} = (0, 1) \), the boolean formula \( \left( (s'_0 + s'_1) \cdot \overline{s'_0 + s'_1} \right) \cdot \overline{s'_1 \cdot s'_0} \) has to be satisfied. It can be simplified to \( \left( s'_0 + s'_1 \right) = (1, 0) = 0_{\text{orr}} \) and \( s'_0 = s'_1 = (1, 0) = 0_{\text{orr}} \). This solution corresponds to the initial state of the latches \( s_0 \) and \( s_1 \) and shows that a state can be reached within one transition step that violates the property.

This example shows that the transition representation of the relation-oriented approach is coarser than that of the function-oriented approach. This kind of different approximation could be useful, since different blackbox scenarios may require different approximation schemes. This is shown by the experimental results presented in the next section.

**V. Experimental Result**

For the experimental comparison of the relation-oriented and function-oriented approach, we have implemented both approaches in the BMX-tool (bounded model checking using 01X-logic) that was already used in the relation-oriented style in [3], [4], [5]. As benchmark examples we used two
model checking examples from the VIS-benchmark suite [22], PicoJava/biu and s1269, whereby both circuits were modified to contain blackboxes. Parts of these modified benchmarks are already established as QBF-variants in the competitive QBF-evaluation that takes place annually [23], [24], [25], [26]. The blackbox circuit designs of PicoJava/biu and s1269 are available in different flavors, such that one, two, or three blackboxes are contained in the partial circuit design that cover 5%, 10%, or 20% of the complete circuit design. Additionally, an error is introduced in the complete circuit such that at least one of the corresponding specification properties is violated—this assures that it is in principal possible to detect errors via 01X-BMC. Given a fixed number of blackboxes and the covered area, we have introduced 15 errors for each circuit and combined them with 10 blackbox constellations, that do not cover the injected error, leading to 150 examples. PicoJava/biu comes with one specification property and s1269 with 5. Since especially for s1269 the introduced error does not violate all properties, there are different numbers of examples available, see column ‘#benchmarks’ in the following tables. For a more detailed description of these modified VIS-benchmarks, see [3], [4], [5].

Another example that we have analyzed is a VLIW-ALU that is described in more detail in [27], [16], see Figure 3. The VLIW-ALU consists of 4 functional units whereby the fourth unit has an error due to an incorrect implementation of the XOR function (the OR function is computed instead). The VLIW-ALU is configurable in its word width, thus enabling us to scale the word width 64. Table V gives details regarding the SAT-solver performance and the AIG synthesis resources for both the VIS-examples and the VLIW-ALU. Table VI gives details for the VLIW-ALU of the function-oriented approach are higher than for the relation-oriented one. Additionally, the time resources for the AIG-synthesis for the function-oriented approach than for the relation-oriented approach. As one can see from the column ‘MiniSat’, the SAT-instances are getting much more complex and hence does not suffer from its coarser transition approximation. Although the number of required AIG-nodes is larger for the relation-oriented approach, the total CPU runtime is much less, namely 11 seconds compared to 39 seconds of the function-oriented approach. As one can see from the column ‘MiniSat’, the SAT-instances are getting much more complex for the function-oriented approach than for the relation-oriented approach.

For the VLIW-ALU, Table IV contains the results. The counterexample has depth 4 for each value of the word width that ranges from 2 to 64. Hence, a BMC-run consists of 3 unsatisfiable and 1 satisfiable SAT-instances. Interestingly, the relation-oriented approach is able to detect the error and hence does not suffer from its coarser transition approximation. Although the number of required AIG-nodes is larger for the relation-oriented approach, the total CPU runtime is much less, namely 11 seconds compared to 39 seconds of the function-oriented approach. As one can see from the column ‘MiniSat’, the SAT-instances are getting much more complex for the function-oriented approach than for the relation-oriented approach. Additionally, the time resources for the AIG-synthesis for the function-oriented approach are higher than for the relation-oriented approach. As one can see from the column ‘MiniSat’, the SAT-instances are getting much more complex for the function-oriented approach than for the relation-oriented approach.
but the relational approach is able to provide the result faster, since it considered less CNF variable assignments compared to the functional approach.

Overall, our experimental analysis shows that the function-oriented approach is able to detect more errors than the relation-oriented approach, along with a higher performance in terms of CPU runtime. But the VLIW-ALU examples show that the relation-oriented approach is able to outperform the function-oriented approach, showing that its coarser transition approximation is helpful for relevant examples.
TABLE V
SAT-SOLVER STATISTICS

<table>
<thead>
<tr>
<th>Group</th>
<th>Property</th>
<th>#Dec.</th>
<th>#Impl.</th>
<th>#Confl.</th>
<th>Time</th>
<th>#Dec.</th>
<th>#Impl.</th>
<th>#Confl.</th>
<th>Time</th>
<th>#Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>b001-p005</td>
<td>property 1</td>
<td>11842</td>
<td>938368</td>
<td>929</td>
<td>0.48</td>
<td>4128</td>
<td>114388</td>
<td>784</td>
<td>0.03</td>
<td>120</td>
</tr>
<tr>
<td>b001-p010</td>
<td>property 2</td>
<td>5782</td>
<td>954169</td>
<td>479</td>
<td>0.42</td>
<td>1678</td>
<td>92941</td>
<td>331</td>
<td>0.02</td>
<td>120</td>
</tr>
<tr>
<td>b001-p020</td>
<td>property 3</td>
<td>18613</td>
<td>1250797</td>
<td>1448</td>
<td>0.52</td>
<td>9327</td>
<td>151448</td>
<td>1426</td>
<td>0.03</td>
<td>120</td>
</tr>
<tr>
<td>b002-p010</td>
<td>property 4</td>
<td>5623</td>
<td>959032</td>
<td>450</td>
<td>0.74</td>
<td>3226</td>
<td>155448</td>
<td>563</td>
<td>0.05</td>
<td>150</td>
</tr>
<tr>
<td>b003-p020</td>
<td>property 5</td>
<td>565</td>
<td>984657</td>
<td>118</td>
<td>0.80</td>
<td>1325</td>
<td>164077</td>
<td>486</td>
<td>0.04</td>
<td>140</td>
</tr>
<tr>
<td>VLIW-ALU 48</td>
<td>property XOR</td>
<td>7420</td>
<td>1025457</td>
<td>360</td>
<td>0.34</td>
<td>121343</td>
<td>1738296</td>
<td>9456</td>
<td>5.73</td>
<td></td>
</tr>
<tr>
<td>VLIW-ALU 64</td>
<td>property XOR</td>
<td>15334</td>
<td>1710010</td>
<td>643</td>
<td>0.67</td>
<td>327550</td>
<td>39277662</td>
<td>16319</td>
<td>16.49</td>
<td></td>
</tr>
</tbody>
</table>

TABLE VI
VLIW-ALU 64 STATISTICS: 3 UNSAT, 1 SAT

<table>
<thead>
<tr>
<th>Depth</th>
<th>AIG Size</th>
<th>Sat Time</th>
<th>AIG Size</th>
<th>Sat Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19908</td>
<td>0.000</td>
<td>8156</td>
<td>0.000</td>
</tr>
<tr>
<td>2</td>
<td>36957</td>
<td>0.028</td>
<td>22283</td>
<td>0.017</td>
</tr>
<tr>
<td>3</td>
<td>54006</td>
<td>0.347</td>
<td>36954</td>
<td>6.427</td>
</tr>
<tr>
<td>4</td>
<td>71055</td>
<td>0.335</td>
<td>51689</td>
<td>11.690</td>
</tr>
</tbody>
</table>

As a summary, both approaches together build an efficient toolbox to analyze partial circuit designs via BMC.

VI. CONCLUSIONS
In this work we have analyzed a relation-oriented and a function-oriented approach for bounded model checking of partial circuit designs. It turned out that the function-oriented approach is much more accurate and has a higher performance for a large set of examples. The relation-oriented approach may also be applied in cases where its coarser transition approximation is accurate enough to detect errors. Hence, both approaches complement one another and allow a flexible application of bounded model checking depending on the circuit under verification.

Acknowledgments: We'd like to thank Christoph Scholl and Christian Miller for fruitful discussions.

REFERENCES