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|herbstri|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 to express that the (propositional) value of a signal is unknown. By adapting BMC to 01X-logic (01X-BMC), it is then possible to detect errors even if the implementation of the circuit design is not finished yet. Hence, 01X-BMC allows for an error analysis already in the early phase of a design implementation. As a consequence, this technique helps to save time and costs and therefore contributes to the optimization of standard circuit design work flows.

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. III. 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). 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 *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 *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, δ, λ) with n primary inputs $x = (x_0, \dots, x_{n-1})$, a register $s = (s_0, \dots, s_{p-1})$ of p latches to store intermediate values, the next-state functions $\delta = (\delta_0, \dots, \delta_{p-1})$ that compute the next-state values of the latches, and the output functions $\lambda = (\lambda_0, \dots, \lambda_{q-1})$ for computing the values at the primary outputs. The latches s 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 δ_i, λ_j is a boolean function $\mathbf{B}^p \times \mathbf{B}^n \to \mathbf{B}$.

Let x^i (s^i) 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 $s_l^{i+1} = \delta_l(s^i, x^i)$. With this we can define a transition relation $T(s^i, x^i, s^{i+1}) = \bigwedge_{l=0}^{p-1} \left(s_l^{i+1} \equiv \delta_l(s^i, x^i)\right)$ which is true iff there is a transition from state s^i to state s^{i+1} by applying input x^i.

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:

$$\mathrm{BMC}(k) = I(s^0) \cdot T(s^0, x^0, s^1) \cdot \dots \cdot T(s^{k-1}, x^{k-1}, s^k) \cdot \overline{P(s^k)}. \qquad (1)$$

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 $\mathrm{BMC}(k)$, 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.

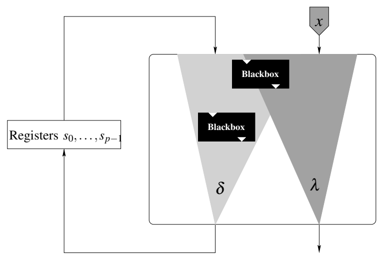


Fig. 1. Sequential circuit with combinational blackboxes.

TABLE I
AND, OR, AND NOT EXTENDED TO 01X-LOGIC.

a	b	$\mathrm{AND}_{01\mathrm{X}}(a,b)$	$\mathrm{OR}_{01\mathrm{X}}(a,b)$	$\mathrm{NOT}_{01\mathrm{X}}(a)$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
0	X	0	1	1
X	0	0	1	X
1	1	1	1	0
1	X	X	X	0
X	1	X	X	X
X	X	X	X	X

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 $0_{01\mathrm{X}} := (1,0), 1_{01\mathrm{X}} := (0,1)$, and $\mathrm{X}_{01\mathrm{X}} := (0,0)$, resp., and the basic operators are adapted, now using tuples (a_0, a_1) and (b_0, b_1) for 01X-variables, as $\mathrm{AND}_{01\mathrm{X}}((a_0,a_1),(b_0,b_1)) := (a_0 + b_0, a_1 \cdot b_1), \mathrm{OR}_{01\mathrm{X}}((a_0,a_1),(b_0,b_1)) := (a_0 \cdot b_0, a_1 + b_1)$, and $\mathrm{NOT}_{01\mathrm{X}}((a_0,a_1)) := (a_1, a_0)$.

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

$$\begin{aligned} T^k(s^0, x^0, \dots, x^{k-1}, s^k) &:= \bigwedge_{i=1}^{k} T(s^{i-1}, x^{i-1}, s^i) \\ &= \bigwedge_{i=1}^{k} \left(s^i \equiv \delta(s^{i-1}, x^{i-1})\right) = \bigwedge_{i=1}^{k} \bigwedge_{j=0}^{p-1} \left(s_j^i \equiv \delta_j(s^{i-1}, x^{i-1})\right). \end{aligned} \qquad (2)$$

For the functional approach, instead, we make use of the *k-step transition function* $\delta_l^k : \mathbf{B}^p \times (\mathbf{B}^n)^k \to \mathbf{B}$ that is inductively defined for a latch l as follows:

$$\begin{array}{rcl} \delta_l^0(s^0) & = & s_l^0 \\ \delta_l^k(s^0,x^0,\ldots,x^{k-1}) & = & \delta_l\big(\delta_0^{k-1}(s^0,x^0,\ldots,x^{k-2}), \\ & & \quad \ldots, \\ & & \quad \delta_{p-1}^{k-1}(s^0,x^0,\ldots,x^{k-2}), \\ & & \quad x^{k-1}\big), \end{array} \tag{3}$$

i. e., the input $s_{l'}^{k-1}$ is substituted by the $(k-1)$-step transition function of latch l'. Finally, we collect all local k-step transition functions within a vector δ^k:

$$\begin{array}{rcl} \delta^k(s^0,x^0,\ldots,x^{k-1}) & := & \big(\delta_0^k(s^0,x^0,\ldots,x^{k-1}), \\ & & \quad \ldots, \\ & & \quad \delta_{p-1}^k(s^0,x^0,\ldots,x^{k-1})\big) \end{array} \tag{4}$$

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

$$\mathrm{BMC}^{\mathrm{r}}(k) \quad := \quad I(s^0)\cdot T^k(s^0,x^0,\ldots,x^{k-1},s^k)\cdot\overline{P(s^k)} \tag{5}$$

$$\mathrm{BMC}^{\mathrm{f}}(k) \quad := \quad I(s^0)\cdot\overline{P\big(\delta^k(s^0,x^0,\ldots,x^{k-1})\big)} \tag{6}$$

are satisfiable iff there exists a counterexample of length k that falsifies property p. The formula $\mathrm{BMC}^{\mathrm{r}}$ ($\mathrm{BMC}^{\mathrm{f}}$) 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 $\mathrm{BMC}^{\mathrm{r}}$ and $\mathrm{BMC}^{\mathrm{f}}$, 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 $\mathtt{AG}(\overline{s_0}\cdot\overline{s_1})$, i. e., none of both latches should ever be 1. The initial state of our system is given by $s_0=0$ and $s_1=0$ and hence the initial state predicate is $I(s^0)=\overline{s_0^0}\cdot\overline{s_1^0}$. The state transition functions at time step i are given as

$$\delta_0(s^i,x^i)=s_0^i+s_1^i+x^i \qquad \text{and} \qquad \delta_1(s^i,x^i)=1. \tag{7}$$

Furthermore, the property to verify at time step i is constituted by $P(s^i)=\overline{s_0^i}\cdot\overline{s_1^i}$ or equivalently $\overline{P(s^i)}=s_0^i+s_1^i$. 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:

$$\begin{array}{rcl} T(s^{i-1},x^{i-1},s^i) & = & \big(s_0^i\equiv\delta_0(s^{i-1},x^{i-1})\big)\cdot\big(s_1^i\equiv\delta_1(s^{i-1},x^{i-1})\big) \\ & = & \big(s_0^i\equiv(s_0^{i-1}+s_1^{i-1}+x^{i-1})\big)\cdot\big(s_1^i\equiv 1\big) \\ & = & (\overline{s_0^i}+s_0^{i-1}+s_1^{i-1}+x^{i-1})\cdot(s_0^i+\overline{s_0^{i-1}})\cdot \\ & & (s_0^i+\overline{s_1^{i-1}})\cdot(s_0^i+\overline{x^{i-1}})\cdot s_1^i. \end{array} \tag{8}$$

To be able to decide satisfiability of formula (5) using a SAT-solver we have to turn the formula into CNF, i. e., into a conjunction of disjunctions of literals. In general, the transformation into an equivalent CNF using the laws of boolean algebra can cause an exponential blow-up of the size of the formula. Hence, for more complex formulas, the solution is to apply the so-called Tseitin-transformation [21] that introduces additional variables for internal sub-formulas. The Tseitin-transformation preserves satisfiability and results in a CNF whose size is linear in the size of the original formula. Putting the parts together, we obtain the following BMC formula for $k=1$:

$$\begin{array}{ll} \mathrm{BMC}^{\mathrm{r}}(1)=\overline{s_0^0}\cdot\overline{s_1^0}\cdot & \rhd\, I(s^0) \\ \qquad (\overline{s_0^1}+s_0^0+s_1^0+x^0)\cdot(s_0^1+\overline{s_0^0})\cdot & \\ \qquad (s_0^1+\overline{s_1^0})\cdot(s_0^1+\overline{x^0})\cdot s_1^1\cdot & \rhd\, T(s^0,x^0,s^1) \\ \qquad (s_0^1+s_1^1). & \rhd\, \overline{P(s^1)} \end{array}$$

Now we construct the BMC-formula using the *functional* approach according to Equation (6). Applying this to our example circuit in Figure 2(a), we obtain the following formula:

$$\mathrm{BMC}^{\mathrm{f}}(1)=(\overline{s_0^0}\cdot\overline{s_1^0})\cdot\big((s_0^0+s_1^0+x^0)+1\big)=(\overline{s_0^0}\cdot\overline{s_1^0}). \tag{9}$$

Both $\mathrm{BMC}^{\mathrm{r}}(1)$ and $\mathrm{BMC}^{\mathrm{f}}(1)$ are satisfiable which means that a counterexample is found and hence the circuit design is erroneous regarding the specification. We mention without proof that both approaches are equivalent regarding the ability to detect counterexamples for *complete* circuit designs. This is not astonishing, since the relational approach is set on top of the functional approach by introducing additional variables for the states at time step i. Interestingly, this issue changes when turning to 01X-BMC of partial circuit designs.

IV. BMC of Blackbox Designs: Relational versus Functional Transition Representation

In this section we investigate the application of a relation-oriented and a function-oriented transition representation in the context of 01X-BMC of partial circuit designs. We show by a small example that the relation-oriented approach is less accurate than the function-oriented approach.

A. Accuracy Deficiency of Relational Representation

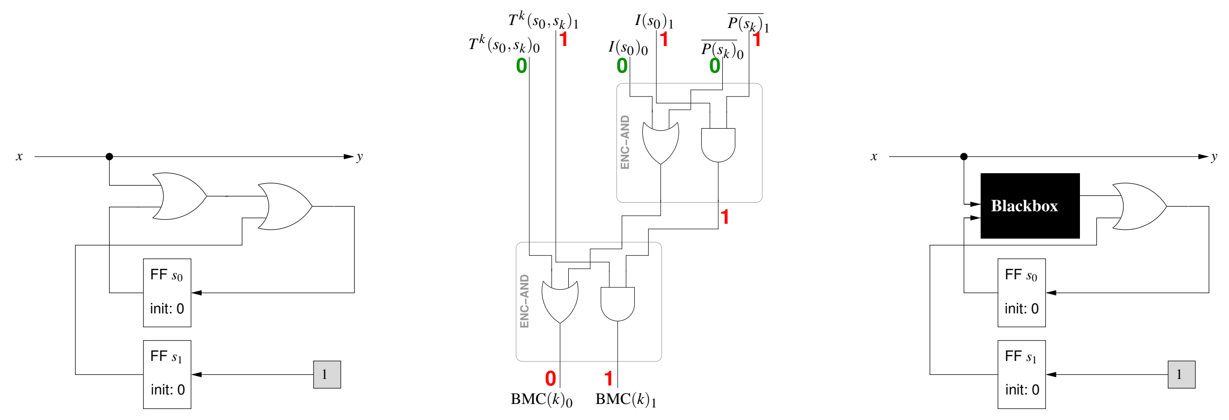
We are now going to adapt the relational transition representation to 01X-BMC of partial circuit designs. Hence, for building the transition relation, we equate next-state-functions $\delta_i(s,x)$ with next-state variables s_i',

$$s_i'\equiv\delta_i(s,x), \tag{10}$$

by using the equivalence-operator $\equiv$, which itself can be expressed by negating the boolean difference. Hence, (10) becomes $\overline{s_i'\oplus\delta_i(s,x)}$, which can be written as

$$s_i'\cdot\delta_i(s,x)+\overline{s_i'}\cdot\overline{\delta_i(s,x)} \tag{11}$$

using only AND, OR, and NOT. Now we look at the binary encoding of expression (11). We use k instead of s_i' and l instead of $\delta_i(s,x)$, 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,



(a) Sequential circuit without blackboxes.

(b) Scenario showing the problem for the relational approach based on the three-partitioned BMC-formula $T^k(s^0,x^0,\dots,x^{k-1},s^k)\cdot I(s^0)\cdot\overline{P(s^k)}$.

(c) Example circuit with a combinational blackbox.

Fig. 2.

(11) becomes

$$\begin{aligned}
&\text{OR}_{\text{01X}}\Big(\quad \text{AND}_{\text{01X}}\Big((k_0,k_1),(l_0,l_1)\Big),\\
&\qquad\qquad \text{AND}_{\text{01X}}\Big(\text{NOT}_{\text{01X}}\big((k_0,k_1)\big),\ \text{NOT}_{\text{01X}}\big((l_0,l_1)\big)\Big)\Big)\\
=\ &\text{OR}_{\text{01X}}\Big(\quad \text{AND}_{\text{01X}}\big((k_0,k_1),(l_0,l_1)\big),\\
&\qquad\qquad \text{AND}_{\text{01X}}\big((k_1,k_0),(l_1,l_0)\big)\Big)\\
=\ &\text{OR}_{\text{01X}}\Big(\big((k_0+l_0),(k_1\cdot l_1)\big),\big((k_1+l_1),(k_0\cdot l_0)\big)\Big)\\
=\ &\big((k_0+l_0)\cdot(k_1+l_1),(k_1\cdot l_1)+(k_0\cdot l_0)\big)\\
=\ &\big((k_0\cdot k_1+k_0\cdot l_1+k_1\cdot l_0+k_0\cdot l_1),(k_0\cdot l_0+k_1\cdot l_1)\big).
\end{aligned}\tag{12}$$

Assume that $k=\text{X}_{\text{01X}}$ and $l=\text{X}_{\text{01X}}$, i. e., $(k_0,k_1)=(0,0)$ and $(l_0,l_1)=(0,0)$. Substituting these values into (12) results in a value $(0,0)=:\text{X}_{\text{01X}}$. Is this correct? We expect from the equivalence operator that it results in value 1 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 $T(s,x,s')$ is computed by a conjunction of all *local* transition relations $T_l(s,x,s_i')$, i. e.,

$$T(s,x,s') := \bigwedge_{l=0}^{p-1} T_l(s,x,s_l') = \bigwedge_{l=0}^{p-1}\big(s_l'\equiv\delta_l(s,x)\big).\tag{13}$$

Recall that our BMC-formula consists of three parts: $\text{BMC}(k)=I(s^0)\cdot\Big(\bigwedge_{i=0}^{(k-1)}T(s^i,x^i,s^{i+1})\Big)\cdot\overline{P(s^k)}$, 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

$$\begin{aligned}
&\exists s^0\exists x^0\exists s^1\exists s^2\dots\exists s^{k-1}\exists x^{k-1}\exists s^k:\\
&\quad I(s^0)\cdot\Big(\bigwedge_{i=0}^{(k-1)}T(s^i,x^i,s^{i+1})\Big)\cdot\overline{P(s^k)}=1.
\end{aligned}\tag{14}$$

Hence, a satisfying solution φ_{CE} for formula (14) requires that each predicate $I(s^0)$, $T(s^i,x^i,s^{i+1})$, and $\overline{P(s^k)}$ is satisfied for $0\le i<k$. Especially for the transition relation this means that in each unfolding step i *all* local transition relations have to be satisfied, i. e., $T_l(s^i,x^i,s_l^{i+1})=1$. Regarding our binary encoding for 01X-logic, $\text{BMC}(k)$ is encoded into a tuple $\big(\text{BMC}(k)_0,\text{BMC}(k)_1\big)$. Since 1_{01X} is encoded as $(0,1)$, our propositional satisfiability problem for the 01X-based approach, i. e., $\big(\text{BMC}(k)_0,\text{BMC}(k)_1\big)=(0,1)$, results in solving the following propositional problem $\overline{\text{BMC}(k)_0}\cdot\text{BMC}(k)_1=1$. 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 $\big(T^k(s^0,x^0,\dots,x^{k-1},s^k)_0,T^k(s^0,x^0,\dots,x^{k-1},s^k)_1\big)$ of the k-fold unfolding of the transition relation, the value $(0,1)=:1_{\text{01X}}$ is opposed.

Consequently, for each global transition relation $T(s^i,x^i,s^{i+1})$ in unfolding step i, this implies for its binary encoding that $(T_0^i,T_1^i)=(0,1)$ must hold. And by applying this implication argument one more time, we end up that for each local transition relation $T_l(s^i,x^i,s_i^{i+1})$ the following condition must also hold:

$$(T_l(s^i,x^i,s_l^{i+1})_0,T_l(s^i,x^i,s_l^{i+1})_1)=(0,1).\tag{15}$$

Now, by substituting $k:=T_l(s^i,x^i,s_l^{i+1})_0$ and $l:=T_l(s^i,x^i,s_l^{i+1})_1$ in Equation (12) for computing $\overline{T_l(s^i,x^i,s_l^{i+1})_0}\cdot T_l(s^i,x^i,s_l^{i+1})_1$ results in computing

$$\overline{(k_0\cdot l_0+k_0\cdot l_1+k_1\cdot l_0+k_1\cdot l_1)}\cdot(k_0\cdot l_0+k_1\cdot l_1).\tag{16}$$

which can be simplified to

$$\overline{k_0}\cdot k_1\cdot\overline{l_0}\cdot l_1+k_0\cdot\overline{k_1}\cdot l_0\cdot\overline{l_1}.\tag{17}$$

The function of (17) computes a 1 iff $(k_0,k_1)=(0,1)=1_{\text{01X}}$ and $(l_0,l_1)=(0,1)=1_{\text{01X}}$, or $(k_0,k_1)=(1,0)=0_{\text{01X}}$ and $(l_0,l_1)=(1,0)=0_{\text{01X}}$. 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 $k = \mathrm{X}_{01X}$ and $l = \mathrm{X}_{01X}$ we *cannot* guarantee that, e. g., k 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_{01X}. But for our relational approach, we have *abused* the equivalence-operator for *storing* the output of the transition function $\delta_l(s,x)$ in the variable s'_l 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., $s'_l = \delta_l(s,x)$ should be true also when $s'_l = \mathrm{X}_{01X}$ and $\delta_l(s,x) = \mathrm{X}_{01X}$.

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_{01X} to the output of the blackbox. Again, the property $\mathtt{AG}(\overline{s_0}\cdot\overline{s_1})$ is checked, resulting in $P(s^i) = \overline{s^i_0}\cdot\overline{s^i_1}$.

Then we obtain the following one-step transition functions: $\delta_0(s_0,s_1,x) = s_1 + \mathrm{X}_{01X}$ and $\delta_1(s_0,s_1,x) = 1$. Doing so, we get the following relational 01X-BMC formula:

$$\begin{aligned}\mathrm{BMC}^{\mathrm{r}}_{01\mathrm{X}}(1) &= I(s^0_0,s^0_1)\cdot\left(s^1_0 \equiv \delta_0(s^0_0,s^0_1,x^0)\right)\cdot\\ &\quad\left(s^1_1 \equiv \delta_1(s^0_0,s^0_1,x^0)\right)\cdot\overline{P(s^1_0,s^1_1)}\\ &= \left(\overline{s^0_0}\cdot\overline{s^0_1}\right)\cdot\left(s^1_0 \equiv (s^0_1+\mathrm{X}_{01X})\right)\cdot\left(s^1_1\equiv 1\right)\cdot\left(s^1_0+s^1_1\right)\\ &= \left(\overline{s^0_0}\cdot\overline{s^0_1}\right)\cdot\left(s^1_0\cdot(s^0_1+\mathrm{X}_{01X})+\overline{s^1_0}\cdot\left(\overline{s^0_1+\mathrm{X}_{01X}}\right)\right)\cdot\left(s^1_1\cdot 1+\overline{s^1_1}\cdot\overline{1}\right)\cdot\left(s^1_0+s^1_1\right)\\ &= \left(\overline{s^0_0}\cdot\overline{s^0_1}\right)\cdot\left(s^1_0\cdot(s^0_1+\mathrm{X}_{01X})+\overline{s^1_0}\cdot\left(\overline{s^0_1+\mathrm{X}_{01X}}\right)\right)\cdot s^1_1\cdot\left(s^1_0+s^1_1\right).\end{aligned} \tag{18}$$

We set $s^i_j := (s^i_{j,0},s^i_{j,1})$ and apply the Jain-encoding from Section III-B to obtain the following boolean formula.[1]

$$\begin{aligned}\mathrm{BMC}^{\mathrm{r,enc}}_{01\mathrm{X}}(1) &= \mathrm{AND}_{01X}\Big(\\ &\quad\mathrm{AND}_{01X}\left(\mathrm{NOT}_{01X}(s^0_{0,0},s^0_{0,1}),\mathrm{NOT}_{01X}(s^0_{1,0},s^0_{1,1})\right),\\ &\quad\mathrm{OR}_{01X}\Big(\mathrm{AND}_{01X}\left((s^1_{0,0},s^1_{0,1}),\mathrm{OR}_{01X}((s^0_{1,0},s^0_{1,1}),(0,0))\right),\\ &\qquad\mathrm{AND}_{01X}\big(\mathrm{NOT}_{01X}(s^1_{0,0},s^1_{0,1}),\\ &\qquad\quad\mathrm{NOT}_{01X}(\mathrm{OR}_{01X}((s^0_{1,0},s^0_{1,1}),(0,0)))\big)\Big),\\ &\quad(s^1_{1,0},s^1_{1,1}),\mathrm{OR}_{01X}\left((s^1_{0,0},s^1_{0,1}),(s^1_{1,0},s^1_{1,1})\right)\Big)\\ &= \mathrm{AND}_{01X}\Big(\mathrm{AND}_{01X}\left((s^0_{0,1},s^0_{0,0}),(s^0_{1,1},s^0_{1,0})\right),\\ &\quad\mathrm{OR}_{01X}\Big(\mathrm{AND}_{01X}\left((s^1_{0,0},s^1_{0,1}),(0,s^0_{1,1})\right),\\ &\qquad\mathrm{AND}_{01X}\left((s^1_{0,1},s^1_{0,0}),\mathrm{NOT}_{01X}(0,s^0_{1,1})\right)\Big),\\ &\quad(s^1_{1,0},s^1_{1,1}),(s^1_{0,0}\cdot s^1_{1,0},s^1_{0,1}+s^1_{1,1})\Big)\\ &= \mathrm{AND}_{01X}\left((s^0_{0,1}+s^0_{1,1},s^0_{0,0}\cdot s^0_{1,0}),\right.\\ &\quad\mathrm{OR}_{01X}\left((s^1_{0,0}+0,s^1_{0,1}\cdot s^0_{1,1}),(s^1_{0,1}+s^0_{1,1},s^1_{0,0}\cdot 0)\right),\\ &\quad\left.(s^1_{1,0},s^1_{1,1}),(s^1_{0,0}\cdot s^1_{1,0},s^1_{0,1}+s^1_{1,1})\right)\\ &= \mathrm{AND}_{01X}\left((s^0_{0,1}+s^0_{1,1},s^0_{0,0}\cdot s^0_{1,0}),(s^1_{0,0}\cdot(s^1_{0,1}+s^0_{1,1}),s^1_{0,1}\cdot s^0_{1,1}+0),\right.\\ &\quad\left.(s^1_{1,0},s^1_{1,1}),(s^1_{0,0}\cdot s^1_{1,0},s^1_{0,1}+s^1_{1,1})\right)\\ &= \mathrm{AND}_{01X}\left((s^0_{0,1}+s^0_{1,1}+s^1_{0,0}\cdot s^1_{0,1}+s^1_{0,0}\cdot s^0_{1,1},s^0_{0,0}\cdot s^0_{1,0}\cdot s^1_{0,1}\cdot s^0_{1,1}),\right.\\ &\qquad\left.(s^1_{1,0}+s^1_{0,0}\cdot s^1_{1,0},s^1_{1,1}\cdot(s^1_{0,1}+s^1_{1,1}))\right)\\ &= (s^0_{0,1}+s^0_{1,1}+s^1_{0,0}\cdot s^1_{0,1}+s^1_{0,0}\cdot s^0_{1,1}+s^1_{1,0}+s^1_{0,0}\cdot s^1_{1,0},\\ &\qquad s^0_{0,0}\cdot s^0_{1,0}\cdot s^1_{0,1}\cdot s^0_{1,1}\cdot s^1_{1,1}\cdot(s^1_{0,1}+s^1_{1,1}))\\ &= (s^0_{0,1}+s^0_{1,1}+s^1_{0,0}\cdot s^1_{0,1}+s^1_{1,0},\ s^0_{0,0}\cdot s^0_{1,0}\cdot s^1_{0,1}\cdot s^0_{1,1}\cdot s^1_{1,1}).\end{aligned} \tag{19}$$

For Formula (19) to evaluate to $1_{01X} = (0,1)$, we need to check the satisfiability of the following formula:

$$\overline{(s^0_{0,1}+s^0_{1,1}+s^1_{0,0}\cdot s^1_{0,1}+s^1_{1,0})}\cdot s^0_{0,0}\cdot s^0_{1,0}\cdot s^1_{0,1}\cdot s^0_{1,1}\cdot s^1_{1,1}. \tag{20}$$

Applying de Morgan's rule we obtain

$$\overline{s^0_{0,1}}\cdot\overline{\boldsymbol{s^0_{1,1}}}\cdot(\overline{s^1_{0,0}}+\overline{s^1_{0,1}})\cdot\overline{s^1_{1,0}}\cdot s^0_{0,0}\cdot s^0_{1,0}\cdot s^1_{0,1}\cdot\boldsymbol{s^0_{1,1}}\cdot s^1_{1,1}. \tag{21}$$

It is easy to see that Formula (21) is unsatisfiable, since it contains both $\overline{s^0_{1,1}}$ and $s^0_{1,1}$ 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:

$$\mathrm{BMC}^{\mathrm{f}}_{01\mathrm{X}}(1) \quad = \quad (\overline{s^0_0}\cdot\overline{s^0_1})\cdot((s^0_1+\mathrm{X}_{01X})+1). \tag{22}$$

We again set $s^i_j := (s^i_{j,0},s^i_{j,1})$ and apply the Jain-encoding from section III-B to obtain the following formula:

$$\begin{aligned}\mathrm{BMC}^{\mathrm{f,enc}}_{01\mathrm{X}}(1) &= \mathrm{AND}_{01X}\big(\mathrm{NOT}_{01X}(s^0_{0,0},s^0_{0,1}),\mathrm{NOT}_{01X}(s^0_{1,0},s^0_{1,1}),\\ &\qquad\mathrm{OR}_{01X}((s^0_{1,0},s^0_{1,1}),(0,0),(0,1))\big)\\ &= \mathrm{AND}_{01X}\big(\\ &\qquad(s^0_{0,1},s^0_{0,0}),(s^0_{1,1},s^0_{1,0}),(s^0_{1,0}\cdot 0\cdot 0,s^0_{1,1}+0+1)\big)\\ &= (s^0_{0,1}+s^0_{1,1},s^0_{0,0}\cdot s^0_{1,0}).\end{aligned} \tag{23}$$

For the resulting tuple of Formula (23) to evaluate to $1_{01X} = (0,1)$, the boolean formula $\overline{(s^0_{0,1}+s^0_{1,1})}\cdot(s^0_{0,0}\cdot s^0_{1,0})$ has to be satisfied. It can be simplified to $(\overline{s^0_{0,1}}\cdot\overline{s^0_{1,1}}\cdot s^0_{0,0}\cdot s^0_{1,0})$, which is satisfied for $(s^0_{0,0},s^0_{0,1}) = (1,0) = 0_{01X}$ and $(s^0_{1,0},s^0_{1,1}) = (1,0) = 0_{01X}$. 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 Results

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

[1]For the sake of readability, we allow the operators AND_{01X} and OR_{01X} to have an arbitrary number of arguments. This is valid, since both operations are commutative and associative.

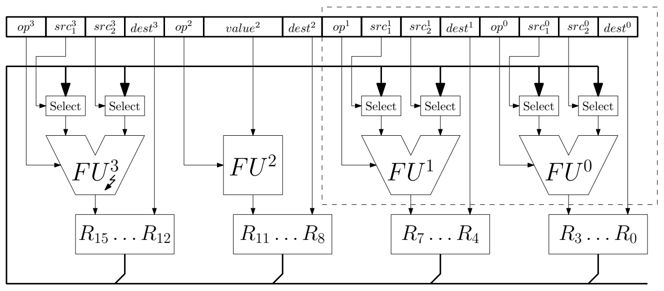


Fig. 3. VLIW-ALU with an error w. r. t. the XOR-implementation in functional unit FU^3.

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.[2] 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 complexity for the underlying decision procedure. To get partial circuit designs, we removed two functional units, for which we were aware that they are unnecessary for the computation of the counterexample, and replaced them by blackboxes.

Tables II and III show the results[3] for the examples related to `PicoJava/biu` and `s1269`. The BMX-tool relies on an internal AIG[4]-representation for the BMC-formula, which is used for applying the binary encoding for 01X-logic. Finally, the resulting AIG is converted to CNF. This CNF is then fed to a SAT-solver, which is MiniSat [30] in our case. The columns denote the CPU runtime that is used for the relation-oriented and the function-oriented approach, resp., detailing the time usage of MiniSat and the remaining time that is spent to build up the AIGs. Note that the relation-oriented approach builds the BMC-formula via conjunction of local transition relations, while the function-oriented approach makes use of a substitution operator as described in Equation (3). Hence, the column 'AIG size' denotes the final number of nodes of the AIG.

As one can see from these results, the function-oriented approach is typically much faster than the relation-oriented approach. This is due to both the time used for building-up the AIG and the time used to solve the final SAT-instance. The SAT-instances are rather easy to solve and although both approaches require less than one second for a benchmark class, the higher performance of the function-oriented approach is clearly visible. But the most interesting point is the number of errors that each approach detects. Regarding this issue, the function-oriented approach is much more accurate, i. e., it detects more errors than the relation-oriented approach, which is due to the differences in accuracy as illustrated in Section IV.

For the examples `PicoJava/biu` and `s1269`, the relation-oriented approach requires in total 223 seconds and detects 603 errors, whereby the function-oriented approach needs 67 seconds and detects 1496 errors. This is, on average, a speed-up of 234% in terms of CPU runtime and an increase in accuracy of 140% compared to the relation-oriented approach. For property 5 of `s1269` there is even a speed-up of 394% and an accuracy increase of 210%. At the bottom of the higher performance of the function-oriented approach is also the number of AIG-nodes that is obviously less than for the relation-oriented approach. E. g., for property 3 of `s1269` the average number of nodes for the final AIG is by a factor of 20 smaller for the function-oriented approach compared to 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 of the function-oriented approach are higher than for the relation-oriented one. Table VI gives details for the VLIW-ALU of word width 64. Table V gives details regarding the SAT-solver resources for both the VIS-examples and the VLIW-ALU.

A reason for this can be found in the properties of the design: Here, many assignments to the primary inputs lead to X values at the output of the transition function. In these cases, the relational approach is unable to generate next-state values so that the equivalence condition in (2) is satisfied and thus, the SAT solver backtracks. In contrast to that, the functional approach tries to find a completion of the current partial assignment of the CNF variables, so that the overall problem is satisfied, which fails.

Both approaches end up with an primary input assignment for which no transition function in any timestep holds the X value,

[2]Our examples are named for instance by `b002-p010`, meaning that the partial circuit design contains 2 blackboxes that cover 10% of the complete circuit design. The percentage is measured in gate equivalents.

[3]The experiments were performed on an AMD Opteron Dual Processor 2.6 Ghz with 4 GB main memory, running a Debian Linux system.

[4]And/Inverter-Graph, see [28], [29].

TABLE II
PicoJava/biu

blackboxes	#benchmarks	time used				#counterexamples		AIG size	
		relational		functional		relational	functional	relational	functional
		MiniSat	Total	MiniSat	Total				
b001-p005	150	0.49	5.59	1.68	6.30	65	71	1,252,378	1,038,000
b001-p010	150	0.31	4.77	0.61	4.06	56	64	1,084,764	711,488
b001-p020	150	0.19	4.68	1.15	4.35	9	50	1,041,910	568,137
b002-p010	150	0.45	7.04	1.11	6.13	15	28	1,628,342	1,052,683
b003-p020	150	0.34	7.24	0.19	4.44	2	6	1,641,324	678,179
Total	750	1.78	29.32	4.74	25.28	147	219	6,648,718	4,048,487

TABLE III
s1269

property	blackboxes	#benchmarks	time used				#counterexamples		AIG size	
			relational		functional		relational	functional	relational	functional
			MiniSat	Total	MiniSat	Total				
1	b001-p005	80	0.64	11.03	0.04	2.35	40	52	2,138,894	227,052
1	b001-p010	40	0.40	6.71	0.05	1.69	11	13	1,314,578	182,611
1	b001-p020	50	0.31	6.26	0.03	1.81	9	16	1,267,785	140,630
1	b002-p010	30	0.21	4.72	0.05	1.44	7	11	931,469	168,962
1	b003-p020	60	0.45	10.68	0.03	2.54	2	12	2,111,596	178,795
1	Total	260	2.01	39.40	0.20	9.83	69	104	7,764,322	898,050
2	b001-p005	90	0.35	7.06	0.01	1.10	40	75	1,356,737	48,946
2	b001-p010	70	0.28	6.41	0.01	1.08	22	52	1,235,167	75,301
2	b001-p020	110	0.49	9.55	0.02	1.83	17	85	1,889,926	73,565
2	b002-p010	80	0.40	8.25	0.03	1.64	15	56	1,602,374	119,745
2	b003-p020	100	0.45	10.68	0.03	2.54	2	12	2,655,956	106,506
2	Total	450	1.97	41.95	0.10	8.19	96	280	8,740,160	424,063
3	b001-p005	20	0.02	0.49	0.01	0.10	11	19	81,283	2,953
3	b001-p010	50	0.17	3.96	0.01	0.74	16	38	761,485	60,294
3	b001-p020	90	0.22	5.51	0.01	1.07	22	66	1,084,248	38,114
3	b002-p010	60	0.11	2.93	0.01	0.60	14	50	567,789	26,929
3	b003-p020	80	0.33	7.96	0.01	1.76	4	44	1,567,361	68,880
3	Total	300	0.85	20.85	0.05	4.27	67	217	4,062,166	197,170
4	b001-p005	70	0.15	3.55	0.01	0.56	34	61	668,256	22,271
4	b001-p010	60	0.21	5.05	0.01	0.90	19	45	976,635	65,861
4	b001-p020	70	0.25	6.11	0.01	1.07	9	47	1,216,228	43,314
4	b002-p010	60	0.22	4.90	0.02	1.11	14	43	956,166	94,579
4	b003-p020	80	0.49	11.39	0.02	2.32	3	30	2,247,070	93,407
4	Totals	340	1.32	26.59	0.07	5.96	79	226	6,064,355	319,432
5	b001-p005	120	0.48	9.95	0.03	1.81	47	99	1,915,729	127,824
5	b001-p010	120	0.42	9.74	0.02	1.74	43	95	1,878,470	109,010
5	b001-p020	120	0.52	10.10	0.03	2.16	23	97	1,981,243	110,250
5	b002-p010	150	0.74	16.25	0.05	3.23	26	102	3,157,955	241,043
5	b003-p020	140	0.80	18.88	0.04	4.18	6	57	3,741,989	189,782
5	Totals	650	2.96	64.92	0.17	13.12	145	450	12,675,386	777,909

TABLE IV
VLIW-ALU, property: XOR-operation

word width	#benchmarks	time used				#counterexamples		AIG size	
		relational		functional		relational	functional	relational	functional
		MiniSat	Total	MiniSat	Total				
2	4 (3 unsat)	0.003	0.05	0.003	0.04	1	1	7,954	5,032
4	4 (3 unsat)	0.006	0.09	0.009	0.08	1	1	13,566	8,712
16	4 (3 unsat)	0.089	0.55	0.265	0.75	1	1	47,238	30,792
24	4 (3 unsat)	0.070	0.90	0.466	1.45	1	1	69,686	45,512
32	4 (3 unsat)	0.112	1.48	0.938	2.39	1	1	92,134	60,232
40	4 (3 unsat)	0.275	2.02	5.469	7.59	1	1	114,582	74,952
48	4 (3 unsat)	0.340	2.66	5.734	8.56	1	1	137,030	89,672
64	4 (3 unsat)	0.674	3.16	16.489	18.63	1	1	181,926	119,112
Total	32 (24 unsat)	1.569	10.91	29.373	39.49	8	8	664,116	434,016

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

Group		relational				functional				#Benchmarks
	Property	#Dec.	#Impl.	#Confl.	Time	#Dec.	#Impl.	#Confl.	Time	
b001-p005	property 1	11842	938368	929	0.48	4128	114838	784	0.03	120
b001-p010	property 2	5782	594166	479	0.42	1678	92941	331	0.02	120
b001-p020	property 3	18613	1250797	1448	0.52	9327	151448	1426	0.03	120
b002-p010	property 4	5623	959032	450	0.74	3226	155448	563	0.05	150
b003-p020	property 5	565	984657	118	0.80	1325	164077	486	0.04	140
VLIW-ALU 48	property XOR	7420	1025457	360	0.34	121343	17382963	9456	5.73	4
VLIW-ALU 64	property XOR	15334	1710010	643	0.67	327550	39277662	16319	16.49	4

TABLE VI

VLIW-ALU 64 STATISTICS: 3 UNSAT, 1 SAT

	relational		functional	
Depth	AIG Size	Sat Time	AIG Size	Sat Time
1	19908	0.000	8186	0.000
2	36957	0.028	22283	0.017
3	54006	0.347	36954	6.427
4	71055	0.335	51689	11.690

As a summary, both approach 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.

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

REFERENCES

[1] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. of Design Automation Conf. (DAC)*, 1999.

[2] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Advances in Computers*, vol. 58, 2003.

[3] M. Herbstritt and B. Becker, "On SAT-based bounded invariant checking of blackbox designs," in *Proc. of Int'l Workshop on Microprocessor Test and Verification (MTV)*, Austin (TX), USA, 2005, pp. 23–28.

[4] M. Herbstritt, B. Becker, and C. Scholl, "Advanced SAT-techniques for bounded model checking of blackbox designs," in *Proc. of Int'l Workshop on Microprocessor Test and Verification (MTV)*, Austin (TX), USA, 2006, pp. 37–44.

[5] M. Herbstritt and B. Becker, "On combining 01X-logic and QBF," in *Proc. of Int'l Conf. on Computer-Aided Systems Theory (EuroCAST)*, ser. LNCS, vol. 4739. Springer, 2007.

[6] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[7] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and Computation*, vol. 98(2), pp. 142–170, 1992.

[8] T. Filkorn, "Functional extension of symbolic model checking," in *Proc. of Int'l Workshop on Computer Aided Verification (CAV)*, ser. LNCS, vol. 575. Springer, 1992, pp. 225–232.

[9] P. Williams, A. Biere, E. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking," in *Proc. of Int'l Conf. on Computer Aided Verification (CAV)*, ser. LNCS, vol. 1855. Springer, 2000, pp. 124–138.

[10] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proc. of Int'l Conf. on Computer Aided Design (ICCAD)*. IEEE CS, 2004, pp. 50–57.

[11] M. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *Proc. of Int'l Conf. on Computer Aided Design (ICCAD)*. IEEE CS, 2004, pp. 510–517.

[12] H.-J. Kang and I.-C. Park, "SAT-based unbounded symbolic model checking," *IEEE Trans. on CAD*, vol. 24, no. 2, pp. 129–140, 2005.

[13] C.-J. H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," *Formal Methods in System Design*, vol. 6, no. 2, pp. 147–189, Mar. 1995.

[14] C.-J. H. Seger, R. B. Jones, J. W. O'Leary, T. F. Melham, M. Aagaard, C. Barrett, and D. Syme, "An industrially effective environment for formal hardware verification." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1381–1405, 2005.

[15] J.-W. Roorda and K. Claessen, "A new SAT-based algorithm for symbolic trajectory evaluation," in *13th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME)*, ser. LNCS, vol. 3725. Springer, 2005, pp. 238–253.

[16] T. Nopper, C. Scholl, and B. Becker, "Computation of minimal counterexamples by using black box techniques and symbolic methods," in *Proc. of Int'l Conf. on Computer Aided Design (ICCAD)*. IEEE CS, 2007.

[17] T. Nopper and C. Scholl, "Approximate symbolic model checking for incomplete designs," in *Proc. of Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, vol. 3312, 2004, pp. 290–305.

[18] ——, "Symbolic model checking for incomplete designs with flexible modeling of unknowns," SFB/TR 14 AVACS, Tech. Rep. 31, July 2007. [Online]. Available: http://www.avacs.org

[19] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 1999.

[20] 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 Symp. (VTS)*, 2000, pp. 263–269.

[21] G. Tseitin, "On the complexity of derivations in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logics*, A. Slisenko, Ed., 1968.

[22] The VIS Group, "VIS: A system for verification and synthesis," in *Proc. of Int'l Conf. on Computer Aided Verification (CAV)*, ser. LNCS, vol. 1102. Springer, 1996, pp. 428–432.

[23] M. Herbstritt, "QBF `blackbox_design` family benchmarks," 2006, [2007-08-29]. [Online]. Available: http://www.qbflib.org/family_detail.php?idFamily=616

[24] ——, "QBF `blackbox-01X-QBF` family benchmarks," 2007, [2007-08-29]. [Online]. Available: http://www.qbflib.org/family_detail.php?idFamily=710

[25] M. Narizzano, L. Pulina, and A. Tacchella, "QBF Evaluation 2006," [2006-08-02]. [Online]. Available: www.qbflib.org/qbfeval

[26] ——, "QBF Evaluation," 2007, [2007-06-08]. [Online]. Available: www.qbflib.org/qbfeval

[27] SFB/TR 14 AVACS - Subproject S1, "The VLIW ALU Benchmark," 2007. [Online]. Available: http://www.avacs.org

[28] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. on CAD*, vol. 21, no. 12, pp. 1377–1394, 2002.

[29] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *Proc. of Design Automation Conf. (DAC)*, 2006, pp. 532–535.

[30] N. Eén and N. Sörensson, "An extensible SAT-solver." in *Proc. of Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, ser. LNCS, vol. 2919. Springer, 2003, pp. 502–518.