AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Symbolic Model Checking for Incomplete Designs with Flexible Modeling of Unknowns

by
Tobias Nopper and Christoph Scholl

AVACS Technical Report No. 31
July 2007
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Christoph Scholl (`scholl@informatik.uni-freiburg.de`).

Symbolic Model Checking for Incomplete Designs with Flexible Modeling of Unknowns[1]

Tobias Nopper and Christoph Scholl

***Abstract*— We consider the problem of checking whether an incomplete design (i.e. a design containing so-called Black Boxes) can still be extended to a complete design satisfying a given property or whether the property is satisfied for all possible extensions.**

Motivated by the fact that well-known model checkers like SMV or VIS produce incorrect results for CTL formulas when handling unknowns by using the programs' nondeterministic signals, we present an approximate, yet sound algorithm to process incomplete designs. The algorithm is flexible in the sense that for every Black Box a different method can be chosen. This permits us to handle less relevant Black Boxes (in terms of the CTL formula) with larger approximation and thus faster, whereas we do not lose important information when the possible effect of more relevant Black Boxes is modeled by more exact methods.

Additionally, we introduce a concept for exact symbolic model checking of incomplete designs containing several Black Boxes with bounded memory.

Finally we give a series of experimental results demonstrating the effectiveness and feasibility of the methods.

***Index Terms*— Symbolic model checking, verification, Black Boxes, incomplete designs, abstraction, approximation, BDDs**

I. Introduction

DECIDING the question whether a circuit implementation fulfills its specification is an essential problem in computer-aided design of VLSI circuits. Growing interest in universities and industry has led to new results and significant advances concerning topics like property checking, state space traversal and combinational equivalence checking.

For proving properties of sequential circuits, Clarke, Emerson, and Sistla presented model checking for the temporal logic CTL [1]. Burch et al. improved the technique by using symbolic methods based on binary decision diagrams [2] for both state set representation and state traversal in [3], [4].

In this paper we will consider how to perform model checking of *incomplete* circuits, i.e., circuits which contain unknown parts, combined into so-called Black Boxes. In doing so, we will address two interesting questions: The question whether it is still possible to replace the Black Boxes by circuit implementations, so that a given property is satisfied ('realizability') and the question whether the property is satisfied for any possible replacement ('validity').

There are three major benefits symbolic model checking for incomplete circuits can provide: First, instead of forcing verification runs to the end of the design process where the design is completed, it rather allows model checking in early stages of design, where parts may not yet be finished, so that errors can be detected earlier. Second, complex parts of a design can be replaced by Black Boxes, simplifying the design, while many properties of the design still can be proven, yet in shorter time. Third, the location of design errors in circuits not satisfying a model checking property can be narrowed down by iteratively masking potentially erroneous parts of the circuit.

Some well-known model checking tools like SMV [4] (resp. NuSMV [5]), and VIS [6] provide the definition of nondeterministic signals (see [7], [8], [9]). At first sight, signals coming from unknown areas can be handled as nondeterministic signals, but we will show that modeling by nondeterministic signals is not capable of answering the questions of realizability ('is there a replacement of the Black Boxes so that the overall implementation satisfies a given property?') or validity ('is a given property satisfied for all replacements of the Black Boxes?') for arbitrary CTL formulas. This approach is even not able to provide approximate solutions for realizability or validity.

Whereas an *exact* solution to the realizability problem for incomplete designs with several Black Boxes (potentially containing an unrestricted amount of memory) is undecidable in general [12], we will present an *approximate* solution to symbolic model checking for incomplete designs. Our algorithm will not give a definite answer in every case, but it is guaranteed to be sound in the sense that it will never give an incorrect answer; it provides proofs of validity and disproofs of realizability. The experimental results given in Sect. VI show the effectiveness and the feasibility of the approximate method.

Our method is based on symbolic representations of incomplete circuits [10] (which will be introduced in this paper as well). Using these representations we provide different methods for approximating the sets of states satisfying a given property φ. One set is an over-approximation of the set of states satisfying the given CTL formula φ for at least one substitution of the Black Boxes and the second set is an under-approximation of the set of states satisfying the formula for all Black Box substitutions. During one run of symbolic model checking we compute both under-approximations and over-approximations of the states satisfying φ and we use them to provide approximate yet sound answers for realizability and validity.

Our approach is able to use different methods for modeling unknowns at the outputs of different Black Boxes within a single model checking run. This permits us to handle less

The authors are with the Department of Computer Science, Albert-Ludwigs-University Freiburg, D-79110 Freiburg i. Br., Germany, (e-mail: nopper@informatik.uni-freiburg.de; scholl@informatik.uni-freiburg.de)

[1]Parts of the article have been presented at DAC 2001 [10] and FMCAD 2004 [11].

relevant Black Boxes (in terms of the CTL formula) with larger approximation and thus faster, whereas we do not lose important information when the possible effect of more relevant Black Boxes is modeled by more exact methods.

We additionally present a concept how to perform exact symbolic model checking under the bounded memory assumption, i.e. for each of the Black Boxes a fixed upper bound on the number of internal states is assumed. Under this bounded memory assumption the method is able to solve realizability and validity questions *exactly*. The algorithm is based on the extraction of the memory out of the Black Boxes and (conceptually) on considering all possible choices for the Black Box instantiations in parallel by means of symbolic methods.

Related Work: The work of Huth et al. [13], which introduced Kripke Modal Transition Systems (KMTSs), comes closest to this approach. Whereas our simplest algorithm can be modeled by using KMTSs, KMTSs are not able to model the fact that the Black Box outputs can not take different values at the same time, while this constraint will be considered by our method (as will be shown below).

Black Boxes in incomplete designs may be seen as Uninterpreted Functions (UIFs) in some sense. UIFs have been used for the verification of pipelined microprocessors [14], where a validity problem is solved under the assumption that both specification and implementation contain the same Uninterpreted Functions. It is important to note that the use of UIFs has to be limited in this approach: the Uninterpreted Function values can not be used for any *computation* of data in the given design — apart from a (conditional) copying of these values, usage of these values as arguments of other Uninterpreted Function symbols and checks for equality. Whereas in [14], [15], [16], [17] a dedicated class of problems for pipelined microprocessors is solved (which is basically reduced to a combinational problem using an inductive argument), we will deal here with arbitrary incomplete sequential circuits and properties given in the full temporal logic CTL.

A related problem is solved in [18] where a Finite State Machine (FSM) is given which interacts with one unknown component (Black Box). In [18] solutions of language equations are used in order to derive the set of all permissible sequential behaviors for the Black Box so that the combined behavior satisfies an external specification. In that work the specification is not given as a CTL formula, but by another FSM.

Outline: The paper is structured as follows: After giving a brief review of symbolic model checking in Sect. II, we will discuss the results of a method that handles Black Boxes by using nondeterministic signal definitions as provided by SMV and VIS, together with the arising problems in Sect. III. In Sect. IV, we will introduce a new algorithm capable of performing sound and approximate symbolic model checking for incomplete circuits. In Sect. V, we will introduce a concept for an exact algorithm to process incomplete designs in which a fixed upper bound on the number of internal states is assumed for each unknown area. Finally, we will give a series of experimental results demonstrating the effectiveness and feasibility of the methods in Sect. VI. Section VII concludes the paper.

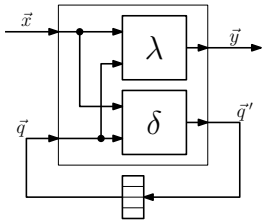


Fig. 1. Mealy automaton.

II. Preliminaries

A. Symbolic Model Checking for Complete Designs

Before we introduce symbolic model checking for incomplete designs we will give a brief review of symbolic model checking for complete designs [3].

Symbolic model checking is applied to Kripke structures which may be derived from sequential circuits on the one hand and to a formula of a temporal logic (in our case CTL (Computation Tree Logic)) on the other hand.

We assume a (complete) sequential circuit to be given by a Mealy automaton

$$M := (\mathbb{B}^{|\vec{q}|}, \mathbb{B}^{|\vec{x}|}, \mathbb{B}^{|\vec{y}|}, \delta, \lambda, \vec{q}^{\,0})$$

with state set $\mathbb{B}^{|\vec{q}|}$, the set of inputs $\mathbb{B}^{|\vec{x}|}$, the set of outputs $\mathbb{B}^{|\vec{y}|}$, transition function $\delta\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \to \mathbb{B}^{|\vec{q}|}$, output function $\lambda\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \to \mathbb{B}^{|\vec{y}|}$ and initial state $\vec{q}^{\,0} \in \mathbb{B}^{|\vec{q}|}$. In the following we will use $\vec{x} = (x_0, \ldots, x_{n-1})$ ($n = |\vec{x}|$) for vectors of input variables, $\vec{y}$ for vectors of output variables, $\vec{q}$ for current state variables and $\vec{q}\,'$ for next state variables. Figure 1 illustrates such a Mealy automaton.

The states of the corresponding Kripke structure are defined as a combination of states and inputs of M. The resulting Kripke structure for M is given by $struct(M) := (S, R, L)$ whereas $S := \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|}$, $R \subseteq S \times S$, $L\colon S \to V$, whereas V is the set of atomic properties $V = \{x_0, \ldots, x_{|\vec{x}|-1}\} \cup \{y_0, \ldots, y_{|\vec{y}|-1}\}$, $R := \big\{\big((\vec{q}, \vec{x}), (\vec{q}\,', \vec{x}\,')\big) \,\big|\, \vec{q}, \vec{q}\,' \in \mathbb{B}^{|\vec{q}|}, \vec{x}, \vec{x}\,' \in \mathbb{B}^{|\vec{x}|}, \delta(\vec{q}, \vec{x}) = \vec{q}\,'\big\}$ and $L\big((\vec{q}, \vec{\epsilon})\big) := \big\{x_i \,\big|\, \epsilon_i = 1\big\} \cup \big\{y_i \,\big|\, \lambda_i(\vec{q}, \vec{\epsilon}) = 1\big\}$.

As usual we write $struct(M), s \models \varphi$ if φ is a CTL formula that is satisfied in state $s = (\vec{q}, \vec{x}) \in S$ of $struct(M)$. If it is clear from the context which Kripke structure is used, we simply write $s \models \varphi$ instead of $struct(M), s \models \varphi$. $\models$ is defined as follows:

$s \models \varphi;\ \varphi \in V \iff \varphi \in L(s)$

$s \models \neg\varphi \iff s \not\models \varphi$

$s \models (\varphi_1 \vee \varphi_2) \iff s \models \varphi_1$ or $s \models \varphi_2$

$s \models EX\varphi \iff \exists s' \in S\colon R(s, s')$ and $s' \models \varphi$

$s \models EG\varphi \iff$ there is a path $(s_0, s_1, s_2, \ldots)$ with $s = s_0$ and $\forall i \geq 0\colon (s_i, s_{i+1}) \in R$ and $s_i \models \varphi$

$s \models E\varphi_1 U \varphi_2 \iff$ there is a path $(s_0, s_1, s_2, \ldots)$ with $s = s_0$ and $\forall i \geq 0\colon (s_i, s_{i+1}) \in R$ and there is a j so that $s_j \models \varphi_2$ and $\forall 0 \leq i < j\colon s_i \models \varphi_1$

a) Fixed point iteration for EG

```
χ_EG(χ_X) {
  old := 1;
  new := χ_X;
  while (old ≠ new) {
    old := new;
    new := χ_X · χ_EX(old);
  }
  return new;
}
```

b) Fixed point iteration for EU

```
χ_EU(χ_X, χ_Y) {
  old := 0;
  new := χ_Y;
  while (old ≠ new) {
    old := new;
    new := χ_Y + (χ_X · χ_EX(old));
  }
  return new;
}
```

Fig. 2. Fixed point iteration algorithms

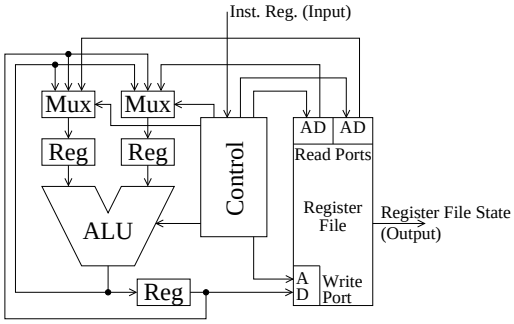


Fig. 3. Pipelined ALU

The remaining CTL operations $\wedge$, EF, AX, AU, AG and AF can be expressed by using $\neg$, $\vee$, EX, EU and EG [4].

In symbolic model checking, sets of states are represented by characteristic functions, which are in turn represented by BDDs. Let $Sat(\varphi)$ be the set of states of $struct(M)$ which satisfy formula φ and let $\chi_{Sat(\varphi)}$ be its characteristic function, then $\chi_{Sat(\varphi)}$ can be computed recursively based on the characteristic function $\chi_R(\vec{q},\vec{x},\vec{q}\,') := \prod_{i=0}^{|\vec{q}|-1}\left(\delta_i(\vec{q},\vec{x}) \equiv q_i'\right)$ of the transition relation R:

$$\begin{aligned}
\chi_{Sat(x_i)}(\vec{q},\vec{x}) &:= x_i\\
\chi_{Sat(y_i)}(\vec{q},\vec{x}) &:= \lambda_i(\vec{q},\vec{x})\\
\chi_{Sat(\neg\varphi)}(\vec{q},\vec{x}) &:= \overline{\chi_{Sat(\varphi)}}(\vec{q},\vec{x})\\
\chi_{Sat((\varphi_1\vee\varphi_2))}(\vec{q},\vec{x}) &:= \chi_{Sat(\varphi_1)}(\vec{q},\vec{x}) + \chi_{Sat(\varphi_2)}(\vec{q},\vec{x})\\
\chi_{Sat(EX\varphi)}(\vec{q},\vec{x}) &:= \chi_{EX}(\chi_{Sat(\varphi)})(\vec{q},\vec{x})\\
\chi_{Sat(EG\varphi)}(\vec{q},\vec{x}) &:= \chi_{EG}(\chi_{Sat(\varphi)})(\vec{q},\vec{x})\\
\chi_{Sat(E\varphi_1U\varphi_2)}(\vec{q},\vec{x}) &:= \chi_{EU}(\chi_{Sat(\varphi_1)},\chi_{Sat(\varphi_2)})(\vec{q},\vec{x})
\end{aligned}$$

with

$$\chi_{EX}(\chi_X)(\vec{q},\vec{x}) := \exists\vec{q}'\exists\vec{x}'\Big(\chi_R(\vec{q},\vec{x},\vec{q}\,')\cdot\big(\chi_X|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}')\Big)$$

χ_{EG} and χ_{EU} can be evaluated by the fixed point iteration algorithms shown in Fig. 2.

A Mealy automaton satisfies a formula φ iff φ is satisfied in all the states of the corresponding Kripke structure which are derived from the initial state $\vec{q}^{\,0}$ of M:

$$\begin{aligned}
M\models\varphi :\Longleftrightarrow\ & \forall\vec{x}\in\mathbb{B}^{|\vec{x}|} : struct(M),(\vec{q}^{\,0},\vec{x})\models\varphi\\
\Longleftrightarrow\ & \big(\forall\vec{x}(\chi_{Sat(\varphi)}|_{\vec{q}=\vec{q}^{\,0}})\big)=1
\end{aligned}$$

B. Realizability and Validity:

Given an incomplete design with Black Boxes and a CTL formula φ, the questions considered in the following are:

1) Is there a replacement of the Black Boxes in the incomplete design, so that the resulting circuit satisfies a given CTL formula φ? If this is true, then the property φ is called *realizable* for the incomplete design.
2) Is a CTL formula φ satisfied for all possible replacements of the Black Boxes? If this is the case, then φ is *valid* for the incomplete design.

III. Model Checking for Incomplete Designs using Nondeterministic Signals

Well-known CTL model checkers such as SMV and VIS provide so-called 'nondeterministic assignments' resp. 'nondeterministic signals' to model nondeterminism [7], [8], [9]. At first sight it appears to be advisable using nondeterministic signals for handling Black Box outputs, since the functionality of Black Boxes is not known. Using nondeterministic signals in [7], [8], [9] has the advantage that they may be handled exactly as primary inputs, leading to a standard CTL model checking procedure for designs containing nondeterministic signals. In this section we motivate our approach by the observation that nondeterministic signals lead to incorrect results when used for CTL model checking of incomplete designs. We will show that they even can not be used to obtain approximate results by analyzing two small examples.

Before doing so, we will report on a larger and more familiar example showing comparable problems. Interestingly, incorrect results of SMV (resp. VIS) due to nondeterministic signals can be observed for the well-known pipelined ALU circuit from [3] (see Fig. 3). In [3], Burch et al. showed by symbolic model checking that (among other CTL formulas) the following formulas are satisfied for the pipelined ALU (the formulas essentially say that the content of the register file $\mathbf{R}$ two (resp. three) clock cycles in the future is uniquely determined by the current state of the system):

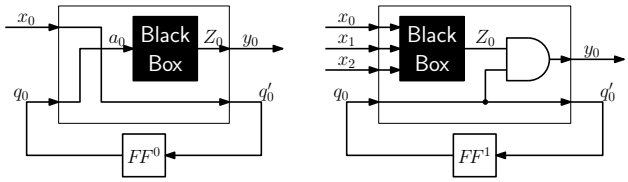
$$AG\big((EX)^2\mathbf{R}\equiv(AX)^2\mathbf{R}\big) \quad (1)$$

$$AG\big((EX)^3\mathbf{R}\equiv(AX)^3\mathbf{R}\big) \quad (2)$$

Now we assume that the ALU's adder has not yet been implemented and it is replaced by a Black Box. The outputs of the Black Box are modeled by nondeterministic signals. In this situation SMV provides the result that formula (2) is not satisfied.[2] However, it is clear that there is at least one replacement of the Black Box which satisfies the CTL formula (a replacement by an adder, of course). Moreover, it is not hard to see, that the formula is even true *for all* possible replacements of the Black Box by any (combinational or sequential) circuit, so one would expect SMV to provide a positive answer both for formula (1) and formula (2).

Obviously, the usage of nondeterministic signals leads to non-exact results. Yet, one might consider that although the results are not exact, they might be approximate in some sense. We will disprove this by analyzing two small exemplary circuits with SMV (similar considerations can be done for VIS as well).

[2]Using VIS, the verification already fails for formula (1) — this is due to a slightly different modeling of automata by Kripke structures in VIS and SMV.



a) First counterexample b) Second counterexample

Fig. 4. Counterexamples

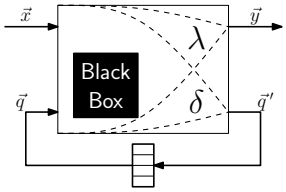


Fig. 5. Mealy automaton with Black Box

Hypothesis 1: 'A negative result of SMV means that a property is not valid.': The circuit from Fig. 4 a) together with formula $\varphi_1 = AG(AXy_0 \vee AX\neg y_0)$ provides us a counterexample for this hypothesis. Formula φ_1 checks whether in all states which are reachable from an initial state the output of the Black Box is the same for all successor states. If we substitute the Black Box output by a nondeterministic signal (modeled in SMV by a new primary input), SMV obviously provides the result that φ_1 is *not* satisfied. Now consider two finite primary input sequences from an initial state which differ only in the last element. Since the Black Box input does not depend on the primary input, but only on the state of the flip-flop (see Fig. 4 a)), these two primary input sequences produce the same input sequence at the Black Box input. Thus, the primary output (which is the same as the Black Box output) will be the same for both input sequences. This means that the CTL formula φ_1 is satisfied for all possible Black Box substitutions, thus it is valid. So we observe that a negative result of SMV does *not* mean that a property is not valid.

Hypothesis 2: 'A negative result of SMV means that a property is not realizable.': We consider the circuit shown in Fig. 4 b) and the CTL formula $\varphi_2 = AGy_0$. We assume that the flip-flop is initialized by 1. If we replace the Black Box output by a nondeterministic signal (modeled internally by a new primary input), SMV provides the result that φ_2 is *not* satisfied. However, it is easy to see that the formula is satisfied if the Black Box is substituted with the constant **1** function; so the property is realizable. Thus, a negative result of SMV does *not* mean that a property is not realizable.

Hypothesis 3: 'A positive result of SMV means that a property is valid.': Again, we consider the example shown in Fig. 4 b) and the CTL formula $\varphi_3 = \neg\varphi_2 = EF\neg y_0$. If we substitute the Black Box output by a nondeterministic signal, SMV provides the result that φ_3 *is* satisfied. However, since property φ_3 is the negation of property φ_2 which has been proven to be realizable when considering the second hypothesis, it is obvious that φ_3 is not valid. Thus, a positive result of SMV does *not* mean that a property is valid.

Hypothesis 4: 'A positive result of SMV means that a property is realizable.': Finally, we reconsider the circuit shown in Fig. 4 a) in combination with $\varphi_4 = \neg\varphi_1 = \neg AG(AXy_0 \vee AX\neg y_0)$. Again, we assume the Black Box output to be a nondeterministic signal and we verify the circuit using SMV, which provides the result that φ_4 *is* satisfied. However φ_4 is not realizable, since $\varphi_4 = \neg\varphi_1$ and φ_1 has been proven to be valid when considering the first hypothesis. Thus, a positive result of SMV does *not* mean that a property is realizable.

Conclusion: Using nondeterministic signals for Black Box outputs is obviously not capable of performing correct model checking for incomplete designs — the approach is even not able to provide an approximate algorithm for realizability or validity.[3]

This motivates our work presented in the next section: We will define an approximate method for proving validity and for falsifying realizability of designs containing Black Boxes. The results are not complete, but they are sound, i.e. depending on the formula and the incomplete design the method may fail to prove validity or falsify realizability, but it will never return incorrect results.

IV. An Approximate Symbolic Model Checking Method for Incomplete Designs with Flexible Handling of Unknowns

A. Flexible Modeling of Black Box Outputs in Symbolic Simulation

For symbolic CTL model checking of a given design, a symbolic representation of its output function λ and of its transition function δ is needed first. In order to generalize CTL model checking to *incomplete* designs (see Fig. 5), the potential effect of the Black Box outputs to the remaining design needs to be modeled in order to compute λ and δ. In [10] we used two different methods, modeling Black Box outputs with differing accuracy: Symbolic $(0,1,X)$-simulation and symbolic Z_i-simulation. Whereas in [10] all Black Box outputs in the design were represented with the same method, we present here a method for flexible modeling of different Black Box outputs by differing methods. This method will be applied later on for our approximate model checking algorithm.

Symbolic $(0,1,X)$-simulation is based on the well-known $(0,1,X)$-simulation [19], [20], [21]. Here the value X represents unknown values due to the unknown functionality of the Black Boxes. If some inputs of a gate are set to X during $(0,1,X)$-simulation, the output is equal to X if and only if there are two different replacements of the X values at the inputs by 0's and 1's, which lead to different outputs of the gate. Fig. 6 b) shows a (conventional) $(0,1,X)$-simulation for the combinational circuit shown in Fig. 6 a).

[3]Yet, there are subclasses of CTL, for which VIS and SMV can provide correct results: Considering ACTL (type A temporal operators only, negation only allowed for atomic propositions), a positive result of SMV/VIS means that the property is valid. Considering ECTL (analogously for E operators), a negative result of VIS means that the property is not realizable; this is not true for SMV due to its implicit universal abstraction of the primary inputs (including primary inputs resulting from nondeterministic signals) at the end of the evaluation.

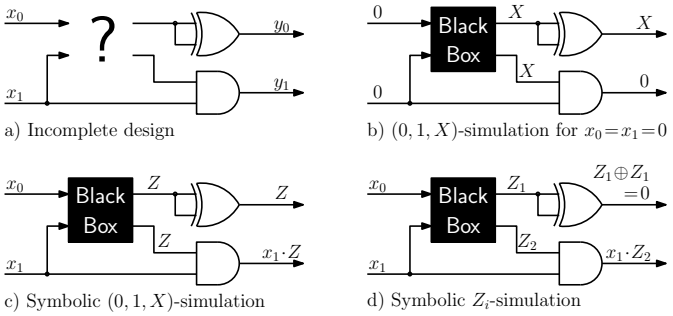


Fig. 6. Different methods to analyze an incomplete design

$(0,1,X)$-simulation may be seen as a (simple) way to over-approximate the set of possible behaviors of systems due to nondeterministic behaviors of their components and in this sense it is a special case of the theory of nondeterministic networks introduced in [22].

For *symbolic* $(0,1,X)$-simulation we introduce a new variable Z, which is used to model the unknown value X of the Black Box outputs. Now, for each output g_i of the incomplete design with primary input variables $x_1,\ldots,x_n$, a BDD representation of g_i is obtained by using a slightly modified version of symbolic simulation [23]. g_i depends on variables $x_1,\ldots,x_n$ and Z and has the following property:

$$g_i|_{\substack{x_1=\epsilon_1 \\ \cdots \\ x_n=\epsilon_n}} = \begin{cases} 1, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\epsilon_1,\ldots,\epsilon_n) \text{ produces } 1 \\ 0, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\epsilon_1,\ldots,\epsilon_n) \text{ produces } 0 \\ Z, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\epsilon_1,\ldots,\epsilon_n) \text{ produces } X \end{cases} \quad (3)$$

See Fig. 6 c) for an example.

To compute BDDs for the functions g_i by symbolic simulation the inputs of the circuit are associated with unique BDD variables as in a conventional symbolic simulation. All output signals of Black Boxes are associated with the new variable Z. Now BDDs for the functions computed by the gates of the circuit are built in topological order treating the Black Box outputs (associated with variable Z) as inputs of the circuit. The gates of the circuit can be processed in a manner similar to a conventional symbolic simulation.[4] When we process an and_2 (or_2) gate, we combine the BDDs for the two predecessor functions by a BDD AND (OR) operation as in the conventional symbolic simulation. For an inv gate we perform a NOT operation on the BDD of the predecessor function, now followed by a *compose* operation (see e.g. [2]) which composes $\overline{Z}$ for Z (written as $g|_{Z\leftarrow\overline{Z}}$ for a composition of $\overline{Z}$ for Z in g).

It is easy to see that this procedure leads to BDD representations fulfilling property 3.

Since $(0,1,X)$-simulation cannot distinguish between unknown values at different Black Box outputs, some information is lost in symbolic $(0,1,X)$-simulation. This problem can be solved at the cost of additional variables: Instead of using the same variable Z for all Black Box outputs, symbolic Z_i-simulation introduces a new variable Z_i for each Black Box output and performs a (conventional) symbolic simulation. Fig. 6 d) shows an example for symbolic Z_i-simulation. (Note that — in contrast to symbolic $(0,1,X)$-simulation in Fig. 6 c) — the first output can now be proven to be constant 0.)

[4]Since all types of gates can be expressed using two-input and_2 gates, two-input or_2 gates and inv gates, we can assume w.l.o.g. that the gates have types and_2, or_2 or inv.

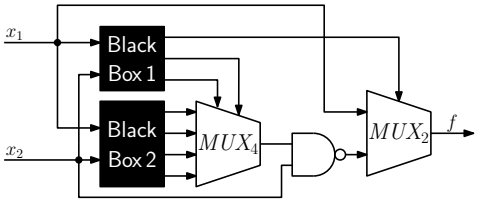


Fig. 7. An exemplary incomplete circuit

We now construct a flexible representation of incomplete circuits which allows some Black Box outputs to be represented as in symbolic $(0,1,X)$-simulation and some Black Box outputs as in symbolic Z_i-simulation: For each output of the Black Boxes, which are to be handled as in symbolic $(0,1,X)$-simulation, we use the variable Z to model the Black Box output, while for each output of the Black Boxes, which are to be handled by symbolic Z_i-simulation we use a new Z_i variable. The simulation now considers the latter Black Box outputs as additional inputs and then performs symbolic $(0,1,X)$-simulation (always replacing Z by $\overline{Z}$ when processing inv gates).

We obtain BDD representations of the circuit outputs g_i with primary input variables $x_1,\ldots,x_n$, (Z_i-simulated) Black Box outputs $Z_1,\ldots,Z_m$ and the Z-variable as inputs:

$$g_i|_{\substack{x_1=\epsilon_1 \\ \cdots \\ x_n=\epsilon_n \\ Z_1=\eta_1 \\ \cdots \\ Z_m=\eta_m}} = \begin{cases} 1, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\epsilon_1,\ldots,\epsilon_n,\eta_1,\ldots,\eta_m) \text{ produces } 1 \\ 0, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\epsilon_1,\ldots,\epsilon_n,\eta_1,\ldots,\eta_m) \text{ produces } 0 \\ Z, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\epsilon_1,\ldots,\epsilon_n,\eta_1,\ldots,\eta_m) \text{ produces } X \end{cases}$$

Example: Figure 7 shows an example: If this circuit is simulated by using symbolic $(0,1,X)$-simulation (meaning that Z is assigned to the outputs of both Black Box 1 and Black Box 2), a total number of 3 variables are needed (x_1,x_2,Z) and the resulting function for the output is $f_Z = Z$.

If the circuit is simulated by using symbolic Z_i-simulation instead (meaning that for each output of Black Box 1 and Black Box 2 a new Z_i variable is used), 9 variables are needed $(x_1,x_2,Z_1,\ldots,Z_7)$, and the function for the output is $f_{Z_i} = \overline{Z}_1 x_1 + Z_1 \cdot \big(\overline{x}_2 + \neg(\overline{Z}_2\overline{Z}_3 Z_4 + Z_2\overline{Z}_3 Z_5 + \overline{Z}_2 Z_3 Z_6 + Z_2 Z_3 Z_7)\big)$ (when variables $Z_1,\ldots Z_7$ are assigned top down to the Black Box outputs appearing in Fig. 7).

When using the flexible method for modeling Black Box outputs, assigning Z to all outputs of Black Box 2, but different Z_i's to the outputs of Black Box 1, e.g., we end up using 6 variables (x_1,x_2,Z,Z_1,Z_2,Z_3) and obtain the function $f_{\text{flex}} = \overline{Z}_1 x_1 + Z_1 \cdot (\overline{x}_2 + Z)$.

So, the flexible method generates an output function that is obviously less complicated than the result of symbolic Z_i-simulation, yet contains more information than the result of

symbolic $(0,1,X)$-simulation. To give an example, for $x_1=1$ and $x_2=0$, the output can be proven to be 1 using the flexible method, while it is not possible to gain this information from symbolic $(0,1,X)$-simulation.

In general, the flexible modeling is at most as exact as symbolic Z_i-simulation, but at least as exact as symbolic $(0,1,X)$-simulation.

B. Symbolic Model Checking for Incomplete Designs

Basic Principle: Symbolic model checking for complete designs computes the set $Sat(\varphi)$ of all states satisfying a CTL formula φ and then checks whether all initial states are included in this set. If so, the circuit satisfies φ.

The situation becomes more complex if we consider incomplete circuits, since for each replacement of the Black Boxes we may have different state sets satisfying φ. In contrast to conventional model checking we will consider two sets instead of $Sat(\varphi)$: The first set is called $Sat_E^{\text{exact}}(\varphi)$ and it contains all states, for which *there is* at least one Black Box replacement so that φ is satisfied. To obtain $Sat_E^{\text{exact}}(\varphi)$ we could *conceptually* consider all possible replacements R of the Black Boxes, compute $Sat^R(\varphi)$ for each such replacement by conventional model checking and determine $Sat_E^{\text{exact}}(\varphi)$ as the union of all these sets $Sat^R(\varphi)$. The second set is called $Sat_A^{\text{exact}}(\varphi)$ and it contains all states, for which φ is satisfied for *all* Black Box replacements. Conceptually, $Sat_A^{\text{exact}}(\varphi)$ could be computed as an intersection of all sets $Sat^R(\varphi)$ obtained for all possible replacements R of the Black Boxes.

Given $Sat_E^{\text{exact}}(\varphi)$ and $Sat_A^{\text{exact}}(\varphi)$, it is easy to prove validity and to falsify realizability for the incomplete circuit: If all initial states are included in $Sat_A^{\text{exact}}(\varphi)$, then all initial states are included in $Sat^R(\varphi)$ for each replacement R of the Black Boxes and thus, φ is satisfied for all replacements of the Black Boxes ("φ is valid"). If there is at least one initial state not belonging to $Sat_E^{\text{exact}}(\varphi)$, then this initial state is not included in $Sat^R(\varphi)$ for all replacements R of the Black Boxes and thus, there is no replacement of the Black Boxes so that φ is satisfied for the resulting complete circuit ("φ is not realizable").

Approximations: For reasons of efficiency we will not compute exact sets $Sat_E^{\text{exact}}(\varphi)$ and $Sat_A^{\text{exact}}(\varphi)$. Instead we will compute *approximations* $Sat_E(\varphi)$ and $Sat_A(\varphi)$ of these sets. To be more precise, we will compute over-approximations $Sat_E(\varphi) \supseteq Sat_E^{\text{exact}}(\varphi)$ of $Sat_E^{\text{exact}}(\varphi)$ and under-approximations $Sat_A(\varphi) \subseteq Sat_A^{\text{exact}}(\varphi)$ of $Sat_A^{\text{exact}}(\varphi)$.

Because of $Sat_E(\varphi) \supseteq Sat_E^{\text{exact}}(\varphi) \supseteq Sat^R(\varphi)$ for arbitrary replacements R of the Black Boxes, we can also guarantee for $Sat_E(\varphi)$ that φ is not realizable if some initial state is not included in $Sat_E(\varphi)$. Analogously we can guarantee that φ is valid if all initial states are included in $Sat_A(\varphi)$ (since $Sat_A(\varphi) \subseteq Sat_A^{\text{exact}}(\varphi) \subseteq Sat^R(\varphi)$).

Approximations of $Sat_E(\varphi)$ and $Sat_A(\varphi)$ will be computed based on an approximate transition relation and on approximate output functions for the corresponding Mealy automaton M. In incomplete designs we have Black Boxes in the functional block defining the transition function δ and the output function λ (see Fig. 5); the approximations of these functions are computed using symbolic simulations as defined in Sect. IV-A.

For this reason there are two approximations of the set of states $Sat(y_i)$ in which the output value y_i of λ_i is true:

- an under-approximation $Sat_A(y_i)$ contains only states in which y_i is true independently from the replacements of the Black Boxes and
- an over-approximation $Sat_E(y_i)$ contains at least all states in which y_i may be true for some replacement of the Black Boxes.

Likewise, there are two types of transitions for the automaton:

- Transitions which exist independently from the replacement of the Black Boxes, i.e. for all possible replacements of the Black Boxes (we will call them 'fixed transitions') and
- transitions which may or may not exist in a complete version of the design — depending on the implementation for the Black Boxes (we will call them 'possible transitions').

We will see later on in this section that for our model checking procedure we will need an over-approximation $\chi_{R_E}(\vec{q},\vec{x},\vec{q}')$ of $\chi_R(\vec{q},\vec{x},\vec{q}')$. $\chi_{R_E}(\vec{q},\vec{x},\vec{q}')$ contains at least all *possible* transitions. An under-approximation $\chi_{R_A}(\vec{q},\vec{x},\vec{q}')$ containing all fixed transitions could be computed as well, however it is not needed for our algorithm.[5]

Based on approximations χ_{R_E}, $Sat_A(y_i)$, and $Sat_E(y_i)$ we will compute the under-approximations $Sat_A(\varphi)$ and over-approximations $Sat_E(\varphi)$ mentioned above for arbitrary CTL formulas φ. At first, we will describe how we compute χ_{R_E}, $Sat_A(y_i)$, and $Sat_E(y_i)$:

For an incomplete circuit, let there be a number of Black Boxes with outputs modeled by Z and some other Black Boxes with outputs modeled by Z_i's. We then apply the method from Sect. IV-A for computing the transition functions and the output functions. Thus, we introduce new variables Z and $\vec{Z}_l = (Z_{l,1}, Z_{l,2}, \dots)$. The symbolic simulation described above now provides symbolic representations of the output functions $\lambda_i(\vec{q},\vec{x},Z,\vec{Z}_l)$ and transition functions $\delta_j(\vec{q},\vec{x},Z,\vec{Z}_l)$.

In standard model checking for complete designs, an atomic property y_i is satisfied for a state $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|}$ if $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\vec{x}=\vec{x}^{\,\text{fix}}} = 1$. Moreover, standard model checking processes transitions between states based on functions δ_j.

Here, a state *definitely* satisfies an atomic property y_i, if y_i is satisfied for *all* possible assignments to Z and $\vec{Z}_l$ and a state *possibly* satisfies an atomic property y_i, if y_i is satisfied for *at least one* possible assignment to Z and $\vec{Z}_l$. Thus, if $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\vec{x}=\vec{x}^{\,\text{fix}}} = 1$ for some state $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|}$, then we know that λ_i is 1 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}})$ into $Sat_A(y_i)$ and $Sat_E(y_i)$.[6] If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\vec{x}=\vec{x}^{\,\text{fix}}} = 0$, then the output λ_i is 0 in this state independently from the replacement

[5] The algorithm presented here improves on a version from [11], in which an under-approximation χ_{R_A} was used.

[6] Remember that $Sat_A(y_i)$ is an under-approximation of the set of states satisfying y_i for all Black Box substitutions and $Sat_E(y_i)$ is an over-approximation of the set of states satisfying y_i for at least one Black Box substitution.

of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}})$ neither into $Sat_A(y_i)$ nor into $Sat_E(y_i)$. In any other case, the value of y_i is unknown in this state and thus we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$. This leads to the following symbolic representations:

$$\chi_{Sat_A(y_i)}(\vec{q},\vec{x}) := \forall Z \forall \vec{Z}_l \big(\lambda_i(\vec{q},\vec{x},Z,\vec{Z}_l)\big), \tag{4}$$

$$\chi_{Sat_E(y_i)}(\vec{q},\vec{x}) := \exists Z \exists \vec{Z}_l \big(\lambda_i(\vec{q},\vec{x},Z,\vec{Z}_l)\big). \tag{5}$$

Likewise, there is a *possible* transition between two states if the transition exists for *at least one* possible assignment to Z and $\vec{Z}_l$. This leads to

$$\chi_{R_E}(\vec{q},\vec{x},\vec{q}') := \Big(\prod\nolimits_{i=0}^{|\vec{q}|-1} \exists Z \exists \vec{Z}_l \big(\delta_i(\vec{q},\vec{x},Z,\vec{Z}_l) \equiv q_i'\big)\Big). \tag{6}$$

An additional improvement of approximations can be obtained by a slightly modified definition of χ_{R_E}:

$$\chi_{R_E}(\vec{q},\vec{x},\vec{q}') := \exists \vec{Z}_l \Big(\prod\nolimits_{i=0}^{|\vec{q}|-1} \exists Z \big(\delta_i(\vec{q},\vec{x},Z,\vec{Z}_l) \equiv q_i'\big)\Big). \tag{7}$$

(Due to the different meaning of variables $\vec{Z}_l$ and variable Z (representing unknowns X) it is easy to see that the order of $\prod$ and $\exists Z$ can not be interchanged in equation (7).)

Based on χ_{R_E}, $Sat_A(y_i)$ and $Sat_E(y_i)$, it is possible to define rules how arbitrary CTL formulas can be recursively evaluated. We show here how to evaluate $Sat_A(EX\psi)$, $Sat_E(EX\psi)$, $Sat_A(\neg\psi)$, $Sat_E(\neg\psi)$, $Sat_A(\psi_1 \vee \psi_2)$, and $Sat_E(\psi_1 \vee \psi_2)$:

For each state $(\vec{q},\vec{x})$, $\chi_{R_E}(\vec{q},\vec{x},\vec{q}')$ gives us the set of $\vec{q}'$ values the possible successors can have. Each of these different $\vec{q}'$ values represents a *set* $S_{\vec{q}'} := \{(\vec{q}',\vec{x}') | \vec{x}' \in \mathbb{B}^{|\vec{x}|}\}$ of possible successor states sharing this $\vec{q}'$ value (yet with arbitrary value of $\vec{x}\,'$). So, if for a state $(\vec{q},\vec{x})$ one of the states in one of these possible successor sets $S_{\vec{q}'}$ possibly satisfies ψ (i.e. is in $Sat_E(\psi)$), the current state possibly satisfies $EX\psi$ and is thus included in $Sat_E(EX\psi)$:

$$\chi_{Sat_E(EX\psi)}(\vec{q},\vec{x}) := \exists \vec{q}' \Big(\chi_{R_E}(\vec{q},\vec{x},\vec{q}') \cdot \exists \vec{x}' \big(\chi_{Sat_E(\psi)}\big|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}')\Big)$$

On the other hand, if in each set $S_{\vec{q}'}$ of possible successors of $(\vec{q},\vec{x})$ there is at least one state that definitely satisfies ψ (i.e. is in $Sat_A(\psi)$), then for each Black Box implementation at least one successor of state $(\vec{q},\vec{x})$ satisfies ψ and thus, the current state $(\vec{q},\vec{x})$ definitely satisfies $EX\psi$ and is included in $Sat_A(EX\psi)$:

$$\chi_{Sat_A(EX\psi)}(\vec{q},\vec{x}) := \forall \vec{q}' \Big(\chi_{R_E}(\vec{q},\vec{x},\vec{q}') \rightarrow \exists \vec{x}' \big(\chi_{Sat_A(\psi)}\big|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}')\Big)$$

Fig. 8 illustrates the sets.

Negation is evaluated as follows: Since $Sat_E(\psi)$ is an over-approximation of all states in which ψ *may be* satisfied for some Black Box replacement, we do know that for an arbitrary state in $\mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \setminus Sat_E(\psi)$ there is no Black Box replacement so that ψ is satisfied in this state or, equivalently, $\neg\psi$ is definitely satisfied in this state for all Black Box replacements.

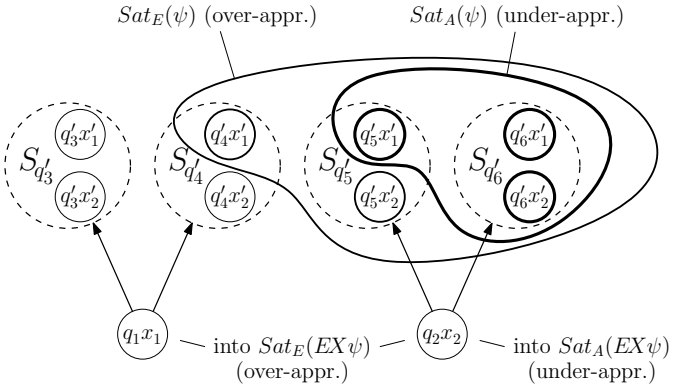


Fig. 8. Evaluation of $Sat_A(EX\psi)$, $Sat_E(EX\psi)$

This means that we can use $\mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \setminus Sat_E(\psi)$ as an under-approximation $Sat_A(\neg\psi)$. Since an analogous argument holds for $Sat_A(\psi)$ and $Sat_E(\neg\psi)$ we define

$$\chi_{Sat_A(\neg\psi)}(\vec{q},\vec{x}) := \overline{\chi_{Sat_E(\psi)}(\vec{q},\vec{x})} \quad \text{and}$$

$$\chi_{Sat_E(\neg\psi)}(\vec{q},\vec{x}) := \overline{\chi_{Sat_A(\psi)}(\vec{q},\vec{x})}.$$

$Sat_A(\psi_1 \vee \psi_2)$ is build by the union of $Sat_A(\psi_1)$ and $Sat_A(\psi_2)$, analogously for $Sat_E(\psi_1 \vee \psi_2)$.

Finally, $\varphi = EG\psi$ and $\varphi = E\psi_1 U \psi_2$ can be evaluated by their standard fixed point iterations (see Fig. 2) based on the evaluation of EX defined above (two separate fixed point iterations for Sat_A and Sat_E). We do not need to define more CTL operations, since other CTL operations can be expressed using the operations discussed so far.

Finally, the result of the recursive computation can be evaluated as follows:[7]

$$\big(\forall \vec{x} (\chi_{Sat_A(\varphi)}|_{\vec{q}=\vec{q}^{\,0}})\big) = 1 \implies \varphi \text{ is valid}$$

$$\big(\exists \vec{x} (\overline{\chi_{Sat_E(\varphi)}}|_{\vec{q}=\vec{q}^{\,0}})\big) = 1 \implies \varphi \text{ is not realizable}$$

C. Including Z_i-Variables into the State Space

A further improvement on the accuracy of the two approximated sets considered above can be obtained by including Z_i-variables assigned to Black Box outputs into the state space.

As a motivation for this, consider the simple CTL formula $EF(y \wedge \neg y)$ for a design in which a Black Box output is directly connected to the primary output y. In every state $(\vec{q},\vec{x})$ both y and $\neg y$ are *possibly* satisfied (depending on the Black Box implementation), but they are not *definitely* satisfied. Thus, the method given in the last section computes the result that $y \wedge \neg y$ is possibly satisfied in every state $(\vec{q},\vec{x})$, but not definitely, and the same result holds for $EF(y \wedge \neg y)$. For this reason the method from Section IV-B is neither able to prove validity nor to falsify realizability for the given incomplete design and the given formula.

However, it is clear that there will be no point in time during the computation where y is simultaneously true and false. Problems of this kind can be solved if we include Z_i-variables assigned to Black Box outputs into the states of the Kripke structure. In this way the according Black Box output

[7] Remember that $\vec{q}^{\,0}$ is the initial state of the circuit.

values Z_i are constant within each single state and therefore in our example y has a fixed value for each state.

Note that it is not always necessary to include *all* Z_i's into the state space; this provides another possibility of flexibly processing the unknowns at this point, which can be used as a tradeoff between efficiency and accuracy.

Let $\vec{Z}_o$ be the Z_i-simulated Black Box outputs that are included into the state space and let $\vec{Z}_l$ be the Z_i-simulated Black Box outputs that are not included. Then the values of $\vec{Z}_o$ are constant within each single state, while the values of $\vec{Z}_l$ are arbitrary as they were before.

Both the output function $\lambda(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)$ and the transition function $\delta(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)$ can be computed by using the symbolic simulation from Sect. IV-A, whereas for symbolic simulation it is not necessary to distinguish between $\vec{Z}_l$ and $\vec{Z}_o$.

We now describe how to compute the sets of states definitely or possibly satisfying the atomic CTL formula y_i:

If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\,\vec{x}=\vec{x}^{\,\text{fix}},\,\vec{Z}_o=\vec{Z}_o^{\,\text{fix}}}=1$ for some state $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}},\vec{Z}_o^{\,\text{fix}})\in\mathbb{B}^{|\vec{q}|\times|\vec{x}|\times|\vec{Z}_o|}$, then we know that λ_i is 1 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}},\vec{Z}_o^{\,\text{fix}})$ into $Sat_A(y_i)$ and $Sat_E(y_i)$.

If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\,\vec{x}=\vec{x}^{\,\text{fix}},\,\vec{Z}_o=\vec{Z}_o^{\,\text{fix}}}=0$, then the output λ_i is 0 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}},\vec{Z}_o^{\,\text{fix}})$ neither into $Sat_A(y_i)$ nor $Sat_E(y_i)$. In any other case, the value of y_i is unknown in this state and thus we include $(\vec{q}^{\,\text{fix}},\vec{x}^{\,\text{fix}},\vec{Z}_o^{\,\text{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$.

Based on this, the set of states definitely satisfying the atomic property y_i can now be computed as follows:

$$\chi_{Sat_A(y_i)}(\vec{q},\vec{x},\vec{Z}_o) := \forall Z\forall\vec{Z}_l\big(\lambda_i(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)\big)$$
$$\chi_{Sat_E(y_i)}(\vec{q},\vec{x},\vec{Z}_o) := \exists Z\exists\vec{Z}_l\big(\lambda_i(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)\big).$$

Analogously, we define the characteristic function of possible transitions:

$$\chi_{R_E}(\vec{q},\vec{x},\vec{Z}_o,\vec{q}') := \Big(\prod_{i=0}^{|\vec{q}|-1}\exists Z\exists\vec{Z}_l\big(\delta_i(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)\equiv q_i'\big)\Big)$$

As for formulas (6) and (7), an additional improvement of approximations can be obtained by

$$\chi_{R_E}(\vec{q},\vec{x},\vec{Z}_o,\vec{q}') := \Big(\exists\vec{Z}_l\prod_{i=0}^{|\vec{q}|-1}\exists Z\big(\delta_i(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)\equiv q_i'\big)\Big).$$

Based on χ_{R_E}, we define $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$ as follows:

$\chi_{R_E}(\vec{q},\vec{x},\vec{Z}_o,\vec{q}')$ gives us the $\vec{q}'$values of possible successors of a state $(\vec{q},\vec{x},\vec{Z}_o)$. Now each of these different $\vec{q}\,'$ values represents a set $S_{\vec{q}'} := \{(\vec{q}\,',\vec{x}\,',\vec{Z}_o')\,|\,\vec{x}\,'\in\mathbb{B}^{|\vec{x}|},\vec{Z}_o'\in\mathbb{B}^{|\vec{Z}_o|}\}$ possible successor states sharing this $\vec{q}\,'$ value. For $Sat_E(EX\psi)$, we include all states $(\vec{q},\vec{x},\vec{Z}_o)$, for which there exists a possible successor set $S_{\vec{q}'}$ in which there is a $\vec{x}\,'$, so that for at least one Black Box output value $\vec{Z}_o'$: $(\vec{q}',\vec{x}',\vec{Z}_o')$ possibly satisfies ψ, i.e.

$$\chi_{Sat_E(EX\psi)}(\vec{q},\vec{x},\vec{Z}_o) :=$$
$$\exists\vec{q}'\Big(\chi_{R_E}(\vec{q},\vec{x},\vec{Z}_o,\vec{q}')\cdot\exists\vec{x}'\exists\vec{Z}_o'\big(\chi_{Sat_E(\psi)}|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'\\ \vec{Z}_o\leftarrow\vec{Z}_o'}}\big)(\vec{q}',\vec{x}',\vec{Z}_o')\Big).$$

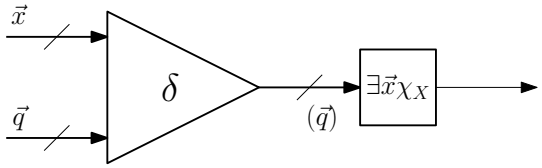


Fig. 9. Illustration for the functional preimage computation for complete designs.

Similarly, for $Sat_A(EX\psi)$, we include all states $(\vec{q},\vec{x},\vec{Z}_o)$, for which in all possible successor sets $S_{\vec{q}'}$ there is a $\vec{x}'$, so that for all Black Box output values $\vec{Z}_o'$: $(\vec{q}',\vec{x}',\vec{Z}_o')$ definitely satisfies ψ, i.e.

$$\chi_{Sat_A(EX\psi)}(\vec{q},\vec{x},\vec{Z}_o) :=$$
$$\forall\vec{q}'\Big(\chi_{R_E}(\vec{q},\vec{x},\vec{Z}_o,\vec{q}')\to\exists\vec{x}'\forall\vec{Z}_o'\big(\chi_{Sat_A(\psi)}|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'\\ \vec{Z}_o\leftarrow\vec{Z}_o'}}\big)(\vec{q}',\vec{x}',\vec{Z}_o')\Big).$$

The computation of all remaining CTL operators $\neg$, EG and EU is performed as described above. The result of the recursive computation can be evaluated as follows:

$$\big(\forall\vec{x}\forall\vec{Z}_o(\chi_{Sat_A(\varphi)}|_{\vec{q}=\vec{q}^{\,0}})\big)=1 \implies \varphi \text{ is valid}$$
$$\big(\exists\vec{x}\forall\vec{Z}_o(\overline{\chi_{Sat_E(\varphi)}}|_{\vec{q}=\vec{q}^{\,0}})\big)=1 \implies \varphi \text{ is not realizable}$$

Obviously, including all Z_i-variables into the state space is one extreme case of the method presented in this section. If we include only a part of the Z_i-variables into the state space, then smaller sets of states and transitions have to be considered, which can lead to a less complex model checking run without necessarily losing the accuracy needed for solving the problem.

D. Functional Preimage Computation

For complete designs, there are two methods to compute the preimage of a given set of states, as it is needed for the computation of $Sat(EX\psi)$ [24], [25]:

So far, we used the *relational* approach in our approximate model checking approach for incomplete designs. For complete designs this approach builds the characteristic function of the transition relation

$$\chi_R(\vec{q},\vec{x},\vec{q}') := \prod\nolimits_{i=0}^{|\vec{q}|-1}\big(\delta_i(\vec{q},\vec{x})\equiv q_i'\big)$$

which is then used in the actual preimage computation for a given set of states (represented by χ_X in this case):

$$\chi_{EX}(\chi_X)(\vec{q},\vec{x}) := \exists\vec{q}'\exists\vec{x}'\big(\chi_R(\vec{q},\vec{x},\vec{q}')\cdot\big(\chi_X|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}')\big)$$

The *functional* approach uses the *compose* operator, defined by $f|_{x_i\leftarrow g} := \overline{g}\cdot f|_{x_i=0} + g\cdot f|_{x_i=1}$ for $f,g\colon\mathbb{B}^n\to\mathbb{B}$ and an input variable x_i of f. Based on the compose operator, the preimage of a set of states given by χ_X can be computed as follows:

$$\chi_{EX}(\chi_X)(\vec{q},\vec{x}) := \big(\exists\vec{x}\chi_X(\vec{q},\vec{x})\big)|_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x})}$$

Note that the number of necessary variables can be decreased by using compose operations instead of transition relations, since the $\vec{q}'$variables are no longer needed. Moreover, the computation of the transition *relation* is not needed. Due to

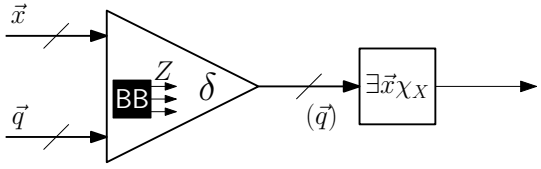


Fig. 10. Illustration for the functional preimage computation for incomplete designs in which the Black Box outputs are modeled by Z.

this, the functional version of preimage computation is often more efficient than the relational version [24].

We now look into the question of how to generalize functional preimage computation so that we can use it for model checking of incomplete designs. In doing so, we (first) confine ourselves to the case where all Black Box outputs are modeled with Z.

To prepare the generalization to incomplete circuits, we first illustrate functional preimage computation for complete designs by a (straightforward) interpretation of the involved BDDs as Boolean circuits as shown in Fig. 9: The BDD for the characteristic function $\big(\exists \vec{x} \chi_X\big)(\vec{q})$ may be interpreted as a (multiplexer) circuit for which variables q_i are replaced by the corresponding transition functions $\delta_i(\vec{q}, \vec{x})$. The result is a characteristic function depending on variables $(\vec{q}, \vec{x})$ which represents the set of states having a successor in the set X (represented by χ_X).

For *incomplete* designs we have to consider the fact that δ now depends on the additional variable Z (which models the unknown value X). Again, we interpret the BDD for the characteristic function $\big(\exists \vec{x} \chi_X\big)(\vec{q})$ as a multiplexer circuit and replace the inputs q_i by the circuit representing the transition function $\delta(\vec{q}, \vec{x}, Z)$ (remember that outputs of the Black Boxes are replaced by variable Z). Now a *symbolic* $(0, 1, X)$*-simulation* of the resulting circuit (compare Fig. 10) produces a function h with the following property:

$$h|_{\substack{\vec{q}=\epsilon_{\vec{q}} \\ \vec{x}=\epsilon_{\vec{x}}}} = \begin{cases} 1, \text{ if state } (\epsilon_{\vec{q}}, \epsilon_{\vec{x}}) \text{ definitely} \\ \quad \text{has a successor in } \chi_X \\ 0, \text{ if state } (\epsilon_{\vec{q}}, \epsilon_{\vec{x}}) \text{ definitely} \\ \quad \text{has no successor in } \chi_X \\ Z, \text{ if state } (\epsilon_{\vec{q}}, \epsilon_{\vec{x}}) \text{ possibly} \\ \quad \text{has a successor in } \chi_X. \end{cases}$$

However, since we already have BDD representations for $\big(\exists \vec{x} \chi_X\big)(\vec{q})$ and $\delta(\vec{q}, \vec{x}, Z)$, we avoid a conversion of the BDDs into circuits followed by a symbolic $(0, 1, X)$-simulation, but we use Fig. 10 only as a conceptual illustration motivating the definition of a (modified) compose operator on BDDs which produces the same result. The compose operator needs to be adjusted to the additional variable Z in order to mimic the behavior of the symbolic $(0, 1, X)$-simulation. As a result we obtain a new *compose-Z* operator for $f\colon \mathbb{B}^n \to \mathbb{B}$ with input variables $x_1, \ldots, x_n$ and $g\colon \mathbb{B}^{n+1} \to \mathbb{B}$ with input variables $x_1, \ldots, x_n$, Z which is based on the following equation:

$$f|^Z_{x_i \leftarrow g} := \overline{g}|_{Z \leftarrow \overline{Z}} \cdot f|_{x_i=0} + g \cdot f|_{x_i=1} \qquad (8)$$

(Note that we have to replace Z by $\overline{Z}$ after negation in this formula just as in the definition of symbolic $(0, 1, X)$-simulation in Sect. IV-A.) Using this compose-Z operator for the BDD representations $\big(\exists \vec{x} \chi_X\big)(\vec{q})$ and $\delta(\vec{q}, \vec{x}, Z)$ we obtain the same result as if we would perform a symbolic $(0, 1, X)$-simulation of the circuit in Fig. 10.

For the special case of the compose-Z operator we can even improve the accuracy of the simple symbolic $(0, 1, X)$-simulation by replacing equation (8) by equation (9):

$$f|^Z_{x_i \leftarrow g} := \overline{g}|_{Z \leftarrow \overline{Z}} \cdot f|_{x_i=0} + g \cdot f|_{x_i=1} + f|_{x_i=0} \cdot f|_{x_i=1} \qquad (9)$$

The last term may seem to be unnecessary at first sight, yet it is easy to see that for $\big(f|_{x_i=0}\big)|_{\vec{x}=\vec{\epsilon}} = 1$, $\big(f|_{x_i=1}\big)|_{\vec{x}=\vec{\epsilon}} = 1$ and $g|_{\vec{x}=\vec{\epsilon}} = Z$ equation (8) results in $\big(f|^Z_{x_i \leftarrow g}\big)|_{\vec{x}=\vec{\epsilon}} = Z$, whereas equation (9) results in $\big(f|^Z_{x_i \leftarrow g}\big)|_{\vec{x}=\vec{\epsilon}} = 1$ which is more exact (contains more accurate information).

Using the *compose-Z* operator, we can now compute $\chi_{Sat_A(EX\psi)}$ and $\chi_{Sat_E(EX\psi)}$: For $\chi_{Sat_A(EX\psi)}$, we include the states for which there is definitely a successor which definitely satisfies ψ (thus is in $\chi_{Sat_A(\psi)}$):

$$\chi_{Sat_A(EX\psi)}(\vec{q}, \vec{x}) := \forall Z \Big(\big(\exists \vec{x} \chi_{Sat_A(\psi)}\big)|^Z_{\vec{q} \leftarrow \delta(\vec{q}, \vec{x}, Z)} \Big)$$

Analogously, for $\chi_{Sat_E(EX\psi)}$, we include the states for which there possibly is a successor possibly satisfying ψ (thus being in $\chi_{Sat_E(\psi)}$):

$$\chi_{Sat_E(EX\psi)}(\vec{q}, \vec{x}) := \exists Z \Big(\big(\exists \vec{x} \chi_{Sat_E(\psi)}\big)|^Z_{\vec{q} \leftarrow \delta(\vec{q}, \vec{x}, Z)} \Big)$$

Functional preimage computation can be easily extended to the cases that some Black Box outputs are modeled by Z_i's and that some of the Z_i-variables are included into the state space in analogy to Sections IV-A and IV-C. Details are omitted here.

Experiments showing advantages of model checking using the compose-Z operator instead of the relational approach will be given in Sect. VI.

V. Exact Symbolic Model Checking for Black Boxes with Bounded Memory

In the last section, we introduced a method to approximate both $Sat_E^{\text{exact}}(\varphi)$, the set of states, for which there is at least one Black Box replacement so that φ is satisfied, and $Sat_A^{\text{exact}}(\varphi)$, the set of states, for which φ is satisfied for all Black Box replacements. Based on these sets, we were able to provide sound results for falsifying realizability and for proving validity of incomplete designs. Yet, it is not possible to provide a result in every scenario due to the approximate nature of our methods.

In this section we will present a concept for an *exact* method under the assumption that there is a fixed upper bound on the number of flip-flops the possible substitutions of the Black Boxes are allowed to have. Due to this 'bounded memory assumption', the number of different Black Box behaviors is finite and thus, it is conceptually possible to compute $Sat^R(\varphi)$ for each possible replacement R of each Black Box. Then, a CTL formula φ is realizable iff there is a replacement R with all initial states lying in $Sat^R(\varphi)$ and a CTL formula φ is valid iff all initial states lie in $Sat^R(\varphi)$ for all possible replacements R.

Note that the Black Box replacements we will consider are not allowed to use any signals other than the ones connected to

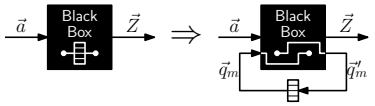


Fig. 11. Extracting flip-flops from a Black Box with bounded memory

the inputs of the Black Box. Thus, the exact method we will present in the following is able to provide an exact answer for the case that the Black Boxes have no *global knowledge* of the surrounding circuit. If the Black Boxes would be allowed to read every signal in the circuit, then we could also use game-theoretic approaches such as [26] for solving the problem. However, considering the small example from Fig. 4 a) together with formula $\varphi_4 = \neg AG(AX y_0 \vee AX \neg y_0)$ (see Sect. III, Hypothesis 4, on page 4) it is easy to see that φ_4 is not realizable (no matter how much memory is used for the Black Box), but the approach from [26] would consider it as realizable due to its implicit assumption that the Black Box behavior may depend on all signals of the circuit.

On the other hand, an explicit approach is obviously not applicable in practice due to the enormous number of possible Black Box substitutions. For that reason we will use symbolic methods to implicitly consider all possible choices for the Black Box substitutions in parallel.

We will first show how Black Boxes with bounded memory can be transformed into combinational Black Boxes, i.e. Black Boxes that may only be substituted by combinational circuits. We will then take a look at a concept for *exact symbolic model checking* for circuits containing one combinational Black Box.

A. Extracting Flip-Flops from a Black Box with Bounded Memory

We consider a Black Box with bounded memory, which means that there is a fixed upper bound on the number of flip-flops the possible substitutions are allowed to have; let m be this upper bound.

Given this assumption, we can separate the flip-flops from the Black Box without changing the behavior: We have to add m additional outputs $\vec{q}'_m$ leading to the flip-flop inputs and m additional inputs $\vec{q}_m$ going back to the Black Box as shown in Fig. 11. The resulting transformed Black Box is combinational, i.e. the possible substitutions are limited to combinational circuits.

Since we can reduce Black Boxes with bounded memory to combinational Black Boxes, it is now sufficient to solve the model checking problem for combinational Black Boxes.

B. A Concept for Exact Symbolic Model Checking of Incomplete Designs with One Combinational Black Box

For the time being, we restrict our view to incomplete circuits containing exactly one combinational Black Box. We showed above that Black Boxes with bounded memory can be reduced to combinational Black Boxes and we will show later how to extend the methods presented here to multiple Black Boxes.

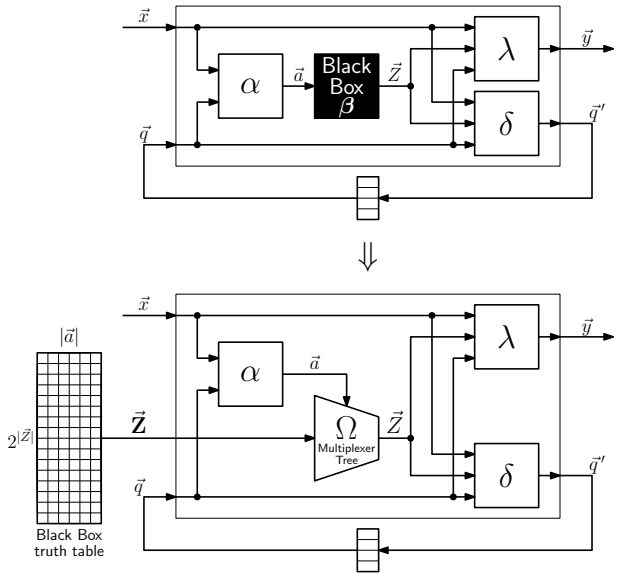


Fig. 12. Incomplete circuit with one combinational Black Box and the modified circuit in which the Black Box has been replaced by its truth table variables and a select function.

Given an incomplete circuit containing exactly one combinational Black Box, we can divide the combinational part of the Mealy automaton into four parts (see upper part of Fig. 12):

Since the Black Box considered in this section is limited to have only combinational substitutions, we can assume the Black Box to compute an unknown boolean function $\beta\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$. Furthermore, let $\alpha\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \to \mathbb{B}^{|\vec{a}|}$ be the boolean function of the circuit part computing the Black Box inputs $\vec{a}$ and $\lambda\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \to \mathbb{B}^{|\vec{y}|}$ resp. $\delta\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \to \mathbb{B}^{|\vec{q}'|}$ be the boolean functions of the circuit parts computing the primary output resp. the next state. While α just depends on the primary input $\vec{x}$ and the current state $\vec{q}$, δ and λ additionally depend on the Black Box outputs $\vec{Z}$. All these functions can be computed using symbolic simulation.

Now we describe how to develop a concept for exact solutions to realizability and validity. To achieve this, we will reduce the question whether there exists a boolean function β so that φ is satisfied (realizability) and the question whether φ is satisfied for all boolean functions β (validity) to existential resp. universal abstraction in propositional logic.

Every function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ can be described by its corresponding truth table with $m \cdot 2^n$ entries; likewise, we can describe the Black Box function $\beta\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$ by a truth table with $|\vec{Z}| \cdot 2^{|\vec{a}|}$ entries.

We consider each entry of this truth table to be a boolean variable $\mathbf{Z}_{i,j} \in \mathbb{B}$ $(0 \le i < 2^{|\vec{a}|},\ 0 \le j < |\vec{Z}|)$. We use $\vec{\mathbf{Z}} := (\mathbf{Z}_{0,0}, \ldots, \mathbf{Z}_{0,|\vec{Z}|-1}, \ldots, \mathbf{Z}_{2^{|\vec{a}|}-1,|\vec{Z}|-1})$ for the whole truth table. An assignment of constant values to variables $\vec{\mathbf{Z}}$ fixes one possible replacement of the (combinational) Black Box. During symbolic model checking the variables $\vec{\mathbf{Z}}$ are included into the state space $(\vec{q}, \vec{x}, \vec{\mathbf{Z}})$. The values of $\vec{\mathbf{Z}}$ do not change during a single run of the resulting system, and thus, fixing the values for $\vec{\mathbf{Z}}$ in an initial state of the system means selecting a certain replacement of the Black Box by a combinational

function.

In order to define both transition function and output function depending on assignments to variables $\vec{\mathbf{Z}}$ we have to introduce a select function $\Omega\colon \mathbb{B}^{|\vec{a}|} \times \mathbb{B}^{(|\vec{Z}|\cdot 2^{|\vec{a}|})} \to \mathbb{B}^{|\vec{Z}|}$ that 'selects' entries from the Black Box truth table variables $\vec{\mathbf{Z}}$ depending on the value of $\vec{a}$ (see lower part of Fig. 12). Formally, $\Omega_i(\vec{a}, \vec{\mathbf{Z}}) := \mathbf{Z}_{a,i}$, whereas a is the integer value described by the binary number $a_{|\vec{a}|-1} \ldots a_1 a_0$. (This select function may be seen as a multiplexer tree.)

Now the output function $\boldsymbol{\lambda}$ and the transition function $\boldsymbol{\delta}$ can be defined using

$$\boldsymbol{\lambda}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) := \lambda\big(\vec{q}, \vec{x}, \Omega\big(\alpha(\vec{q}, \vec{x}), \vec{\mathbf{Z}}\big)\big)$$
$$\text{and} \quad \boldsymbol{\delta}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) := \delta\big(\vec{q}, \vec{x}, \Omega\big(\alpha(\vec{q}, \vec{x}), \vec{\mathbf{Z}}\big)\big).$$

For our exact symbolic model checking, we essentially perform conventional symbolic model checking (see Sect. II) based on $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$ with a state space extended by variables $\vec{\mathbf{Z}}$. Transitions from one state to its successor in this extended state space $(\vec{q}, \vec{x}, \vec{\mathbf{Z}})$ do not change the values assigned to $\vec{\mathbf{Z}}$. This keeps the functionality of the Black Box fixed during an entire run of the system which starts with a certain initial state specifying a constant assignment to $\vec{\mathbf{Z}}$.

Since $\vec{\mathbf{Z}}$ represent the complete truth table of the Black Box β, thus its whole functionality, there is a substitution of β so that a property is satisfied in a certain initial state $(\vec{q}^{\,0}, \vec{x})$ iff there is some assignment to $\vec{\mathbf{Z}}$ so that the property is satisfied in the corresponding state $(\vec{q}^{\,0}, \vec{x}, \vec{\mathbf{Z}})$ of the transformed design (see Fig. 12). Likewise, a property is satisfied in $(\vec{q}^{\,0}, \vec{x})$ for all substitutions of β iff it is satisfied in $(\vec{q}^{\,0}, \vec{x}, \vec{\mathbf{Z}})$ for all possible assignments to $\vec{\mathbf{Z}}$. Thus, after a conventional symbolic model checking (with extended state space $(\vec{q}, \vec{x}, \vec{\mathbf{Z}})$) we can reduce the validity/realizability question to an universal/existential abstraction of $\vec{\mathbf{Z}}$:

$$\big(\forall\vec{\mathbf{Z}}\forall\vec{x}(\chi_{Sat(\varphi)}|_{\vec{q}=\vec{q}^{\,0}})\big) = 1 \iff \varphi \text{ is valid} \tag{10}$$
$$\big(\exists\vec{\mathbf{Z}}\forall\vec{x}(\chi_{Sat(\varphi)}|_{\vec{q}=\vec{q}^{\,0}})\big) = 1 \iff \varphi \text{ is realizable} \tag{11}$$

Example: We check the circuit shown in Fig. 13a with the CTL formula $\varphi = AFy_0$. First, we model the combinational Black Box β by the corresponding $2^1{=}2$ truth table variables $\vec{\mathbf{Z}} = (\mathbf{Z}_0, \mathbf{Z}_1)$ and the select function Ω — in this case, just a multiplexer. The modified circuit is shown in Fig. 13b. We can now compute $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$:

$$\boldsymbol{\lambda}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) = \big(x_0 \oplus q_0 \oplus (\overline{x}_0 \cdot \mathbf{Z}_0 + x_0 \cdot \mathbf{Z}_1)\big)$$
$$\boldsymbol{\delta}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) = \big(q_0 \oplus (\overline{x}_0 \cdot \mathbf{Z}_0 + x_0 \cdot \mathbf{Z}_1)\big)$$

This eventually leads to

$$\chi_{Sat(AFy_0)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) = \big(\mathbf{Z}_0 \equiv (x_0 \equiv q_0)\big) + x_0 \cdot (\mathbf{Z}_0 \oplus \mathbf{Z}_1)$$

Validity and realizability checking:

$$\big(\forall\vec{\mathbf{Z}}\forall\vec{x}(\chi_{Sat(AFy_0)}|_{\vec{q}=\vec{q}^{\,0}})\big) = 0 \iff AFy_0 \text{ is not valid}$$
$$\big(\exists\vec{\mathbf{Z}}\forall\vec{x}(\chi_{Sat(AFy_0)}|_{\vec{q}=\vec{q}^{\,0}})\big) = 1 \iff AFy_0 \text{ is realizable}$$

So, $\varphi = AFy_0$ is satisfied for at least one, but not all Black Box substitutions (more precisely, a substituting inverter causes φ to be satisfied, while all other possible substitutions — constant $\mathbf{0}$ function, constant $\mathbf{1}$ function, wire — do not).

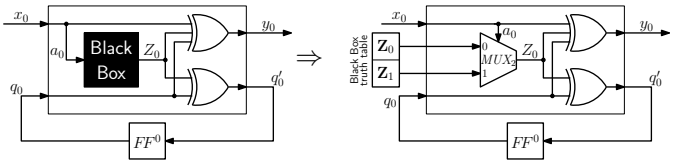


Fig. 13. Example for exact symbolic model checking of incomplete designs with one combinational Black Box.

C. Multiple Black Boxes

It is easy to see that the method presented in this section can be extended to circuits containing multiple Black Boxes by separately replacing them by corresponding truth table variables.

VI. Experimental Results

A. Approximate Model Checking for Incomplete Designs

To demonstrate the feasibility and effectiveness of the presented methods we implemented a model checker that is capable of performing symbolic model checking with flexible modeling of unknowns and exact symbolic model checking. The model checker is based on the BDD package CUDD 2.41 [27] and uses 'Lazy Group Sifting' [28], a reordering technique particularly suited for model checking, and partitioned transition functions [29].

For our experiments we used a class of simple synchronous pipelined ALUs (see Fig. 3) with a register file and a forwarding unit; the circuit is based on the design used in [3]. The ALU itself was able to perform the four logic operations AND, OR, XOR and XNOR as well as the three arithmetic operations ADD, SUB and MUL.

We checked the CTL formula

$$\varphi = AG\Big(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\Big)$$

which corresponds to formula (1) in [3]. It says that whenever the instruction $\mathtt{R}_2{:=}\mathtt{R}_0{\oplus}\mathtt{R}_1$ is given at the inputs, the values in $\mathbf{R}_2$ three clock cycles in the future will be identical to the exclusive-or of $\mathbf{R}_0$ and $\mathbf{R}_1$ in the state two clock cycles in the future ($\mathbf{R}_0$, $\mathbf{R}_1$ and $\mathbf{R}_2$ are the respective first, second and third register in the register file).

All experiments were performed on an Dual Opteron 2GHz with 4GB RAM.

In a first experiment, we inserted an error to the implementation of the XOR operation[8], so it produced incorrect results. We compared a series of complete pipelined ALUs with 16 registers in the register file and varying word width to two incomplete counterparts: For the first, the adder and the multiplier were substituted by Black Boxes and for the second, 12 of the 16 registers in the register file were masked out as well.

It can be seen that property φ is violated for the complete and incomplete designs, independently of the implementation

[8]The lowest bit of the output was the result of an OR instead of an XOR of the two lowest input bits.

word width	No Black Boxes: BDD vars	memory used	BDD nodes	time	Adder and multiplier replaced by Z-assigned Black Boxes: BDD vars	memory used	BDD nodes	time	Adder, multiplier and 12 registers replaced by Z-assigned Black Boxes: BDD vars	memory used	BDD nodes	time
2	117	30049824	201041	8.94	117	15901440	87123	4.54	69	11954432	26374	0.89
4	193	42185472	407339	69.92	193	18527040	101504	8.63	97	11490336	15550	1.00
6	269	73541376	1349311	356.69	269	43841504	115167	15.98	125	14406528	22402	1.54
8	345	238844096	6295929	2780.98	345	47203616	90543	13.89	153	16477632	20557	1.92
12	timeout				497	43848096	83255	24.88	209	26694944	34044	4.79
16	timeout				649	48474368	88473	47.13	265	35660672	28362	5.36
24	timeout				953	44654720	93991	91.29	377	39617536	35514	12.10
32	timeout				1257	53717088	216086	232.21	489	47448224	34562	17.53
48	timeout				1865	61818176	143362	493.07	713	48465760	46259	45.13
64	timeout				2473	64708160	167102	3030.55	937	44284576	46996	82.35

TABLE I

FAULTY PIPELINED ALU WITH 16 REGISTERS:
FALSIFYING THE REALIZABILITY OF $\varphi = AG\big(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$
ASSIGNING Z TO THE BLACK BOX OUTPUTS AND USING TRANSITION RELATIONS.

word width	No Black Boxes: BDD vars	memory used	BDD nodes	time	Adder and multiplier replaced by state space Z_i Black Boxes: BDD vars	memory used	BDD nodes	time	Adder, multiplier and 12 registers replaced by state space Z_i Black Boxes: BDD vars	memory used	BDD nodes	time
2	117	18211616	186320	8.27	121	13469376	49559	2.42	97	13233664	52030	1.81
4	193	43028512	427505	57.26	201	28242464	74669	4.00	153	13768448	57472	4.96
6	269	81169184	1386382	395.44	281	30440352	60754	8.23	209	27957920	70160	4.48
8	timeout				361	40483328	88051	12.90	265	27256864	89714	14.48
12	timeout				521	47836160	116303	33.91	377	34484672	81459	20.46
16	timeout				681	48264384	135068	59.06	489	47732928	98150	35.67
24	timeout				1001	45233152	90236	83.84	713	45776928	113550	66.39
32	timeout				1321	44077024	149996	207.71	937	44598144	91438	83.03
48	timeout				1961	65595168	165290	457.32	1385	50398272	152227	175.17
64	timeout				2601	66618208	182767	2283.81	1833	62326016	160155	287.71

TABLE II

(CORRECT) PIPELINED ALU WITH 16 REGISTERS:
PROVING THE VALIDITY OF $\varphi = AG\big(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$
USING Z_i'S IN THE STATE SPACE FOR ALL BLACK BOX OUTPUTS AND USING TRANSITION RELATIONS.

word width	Outputs of the Black Boxes in Register File modeled with... ...separate Z_i variables *in* the state space, using transition relations: BDD vars	memory used	BDD nodes	time	...separate Z_i variables *not* in the state space, using transition relations: BDD vars	memory used	BDD nodes	time	...one single Z variable *not* in the state space, using transition relations: BDD vars	memory used	BDD nodes	time	...one single Z variable *not* in the state space, using compose-Z: BDD vars	memory used	BDD nodes	time
2	605	578468608	12865546	28945.46	605	24413760	50222	16.80	101	19342304	98152	7.64	67	15842144	85207	3.91
4	1141	628070432	13523750	71524.01	1141	35190400	91815	63.51	133	31807552	86563	7.93	85	16285824	69296	4.22
6	timeout				1677	44675616	115583	114.95	165	35522752	168618	17.43	103	17461920	66970	2.85
8	timeout				2213	54856160	135796	192.02	197	33882944	88538	8.08	121	34211072	102188	7.91
12	timeout				3285	74282144	138534	184.78	261	45648128	151989	26.20	157	30730624	90524	6.02
16	timeout				4357	92998880	155438	257.61	325	48306848	128800	26.52	193	42073920	96041	8.54
24	timeout				6501	216046752	173549	331.88	453	48442432	147198	45.62	265	40262336	105927	16.83
32	timeout				8645	219996512	260208	603.10	581	50662656	102933	60.61	337	48314016	86354	15.18
48	timeout				12933	249238816	355851	725.76	837	45299584	126438	100.89	481	48770368	126000	58.69
64	timeout				17221	564282080	449412	1277.96	1093	55079040	112916	141.08	625	46858432	146922	77.88

TABLE III

(CORRECT) INCOMPLETE PIPELINED ALU WITH 256 REGISTERS: PROVING THE VALIDITY OF
$\varphi = AG\big(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ USING Z_i'S IN THE STATE SPACE FOR THE BLACK BOXES REPLACING THE ADDER AND THE MULTIPLIER AND DIFFERENT METHODS FOR THE BLACK BOXES IN THE REGISTER FILE.

of the adder function, the multiplier function and the registers replaced by Black Boxes.

In Tab. I we give the results for both complete and incomplete pipelined ALUs with varying word width tested with φ. For each word width and each pipelined ALU, the table shows the number of BDD variables ('BDD vars'), the peak memory usage in bytes, the peak number of BDD nodes and the overall time in CPU seconds. The timeout was 12.000 CPU seconds. For this experiment, transition relation based preimage computation was used.

Since multipliers have a large impact on BDD size and thus on computation time, the model checking procedure for complete pipelined ALUs with multipliers of word width beyond 8 bit exceeds the time limit (see Tab. I, columns 2–5).

For the incomplete pipelined ALUs we observed the result that already our weakest method for approximate model checking (using symbolic $(0, 1, X)$-simulated Black Boxes) was able to prove that the property φ is not realizable.

This can be verified for the incomplete pipelined ALUs without adder and multiplier up to a word width of 64 bit within moderate CPU times and moderate memory consumption (see Tab. I, columns 6–9).

The results for the incomplete pipelined ALU, in which most of the register file has been replaced by Black Boxes as well, show a further speedup compared to the complete pipelined ALU (see Tab. I, columns 10–13). This is mainly due to the decrease of needed BDD variables, caused by the reduction of many q_i and q_i' variables to a single Z variable and the simplification of the transition function, which does no longer depend on the input functions of the registers that have been masked out.

Thus, we are able to mask out the most complex parts of the pipelined ALU — the multiplier and the adder — and most of the register file without losing any significance of the result. Note that *all* Black Boxes lie in the cone of influence for this property.

In a second experiment we considered the same CTL formula as above, yet this time we used a *correct* implementation of the XOR operation. In this case, φ is satisfied for the complete and valid for the incomplete pipelined ALUs.

In Tab. II we give the results for both complete and incomplete pipelined ALUs tested with φ. Again, the timeout was 12.000 seconds and preimages were computed using a relation transition.

In this example, the weaker methods assigning Z or non-state-space Z_i's to the Black Box outputs were not powerful enough to prove the validity of φ. However, in all cases the formula could be proven to be valid by assigning Z_i's to the Black Box outputs and including them into the state space.

The number of BDD variables needed for the incomplete pipelined ALU has increased in comparison to symbolic Z-model checking (compare the corresponding columns in Tables I and II); this is due to the use of separate Z_i variables for each Black Box output instead of one single Z variable. The effect can be observed best for the pipelined ALU with partially masked register file. Although slower than the model checking runs in the first experiment, for which all Black Box outputs were modeled with Z, model checking of the incomplete pipelined ALUs with Z_i's in the state space clearly outperforms the conventional model checking of the complete version, for the same reasons as given above.

For a third experiment, we analyzed a pipelined ALU with a larger register file now containing 256 registers. Both the adder and the multiplier of the pipelined ALU were substituted by Black Boxes, and all but the lowest four registers were masked out as well. We again considered the validity of φ.

First, we used separate Z_i-variables for all Black Box outputs, all included into the state space (just as in the second experiment before). We then made use of the flexibility of our method: We reduced the accuracy for the Black Boxes in the register file by removing the according Z_i's from the state space, while keeping the ones for the Black Boxes replacing the adder and the multiplier. In a third series, a further reduction of accuracy for the Black Boxes in the register file was achieved by modeling their outputs with the single variable Z.

Except for the timeouts, we always were able to prove the validity of φ for the incomplete designs.[9]

In Tab. III, columns 2–13, we give the results for the incomplete pipelined ALUs with varying word width tested with φ, using the different methods for the Black Boxes in the register file and Z_i's in the state space for the Black Boxes replacing the adder and the multiplier. Here, the timeout was 86.400 seconds (= 1 day).

If all Black Boxes are modeled with Z_i's in the state space, a complex transition relation has to be build between states that contain a considerable number of Z_i variables, including Z_i variables representing the outputs of registers which were masked out. On account of this, it is only possible to prove validity for a word width up to 4 bit before exceeding the time limit (see Tab. III, columns 2–5).

If only the Z_i's of the Black Boxes masking out the multiplier and the adder are included into the state space, we have to deal with a smaller state space and a less complex transition relation, which leads to the result that we are able to prove validity for all instances within the time limit (see columns 6–9 of Tab. III).

In the case that all Black Box outputs in the register file are modeled using one single Z variable (columns 10–13 of Tab. III), there is a significant decrease in the number of necessary BDD variables. For this reason, there is a further speedup compared to the last experiment and validity could be proven for all bit widths up to 64 within less than 2.5 CPU minutes.

In a last experiment we evaluated the efficiency of our functional preimage computation based on the *compose-Z* operator as introduced in Sect. IV-D. For that purpose we reran the third series of experiments (Z-modeled Black Boxed in the register file and state space Z_i-modeled Black Boxes replacing the adder and the multiplier), now using *compose-Z* for preimage computation. The results are given in columns 14–17 of Tab. III, whereas the corresponding results for preimage computation based on transition relations can be found in columns 10–13.

The results clearly show that the functional approach using *compose*-Z performs even better than the relational approach in this case.

Taken together, the results show that symbolic model checking for incomplete designs with flexible modeling of unknowns is able to provide sound and useful results, yet within shorter time and with fewer memory consumption compared to symbolic model checking for complete designs.

B. Exact Symbolic Model Checking for Black Boxes with Bounded Memory

For a first evaluation of our exact symbolic model checking method that has been presented in Sect. V, we considered a class of arbiters as described in [4]. Given a resource and a number of clients trying to access the resource, the purpose

[9]Whereas reducing the accuracy for the Black Boxes that replace the adder and the multiplier (removing corresponding Z_i's from the state space or replacing them by the single variable Z) would lead to the situation that we are not able to prove validity of φ as already seen in the previous experiment.

of an arbiter is to grant access only to a single client for each clock cycle. An arbiter for n clients has n request inputs $\mathrm{req}_0 \ldots \mathrm{req}_{n-1}$, whereas $\mathrm{req}_i = 1$ iff client i requests access to the resource, and n acknowledge outputs $\mathrm{ack}_0 \ldots \mathrm{ack}_{n-1}$, whereas $\mathrm{ack}_i = 1$ iff the arbiter acknowledges the request of client i.

In [4], three CTL properties are given that an arbiter for n clients must satisfy in order to work as expected:

$$\varphi_1^n = \bigwedge\nolimits_{0 \le i < j < n} \big(AG\neg(\mathrm{ack}_i \wedge \mathrm{ack}_j)\big)$$
$$\varphi_2^n = \bigwedge\nolimits_{0 \le i < n} \big(AGAF(\mathrm{req}_i \rightarrow \mathrm{ack}_i)\big)$$
$$\varphi_3^n = \bigwedge\nolimits_{0 \le i < n} \big(AG(\mathrm{ack}_i \rightarrow \mathrm{req}_i)\big)$$

Property φ_1^n essentially says that no two acknowledge outputs are asserted simultaneously, φ_2^n states that every persistent request should be eventually acknowledged and φ_3^n checks whether no acknowledge is asserted without an according request.

For an arbiter with n clients, [4] provides an implementation that uses $2 \cdot n$ flip-flops.

We now focus on the question whether there is an implementation using less than $2 \cdot n$ flip-flops. For this, we consider an arbiter with n clients as an incomplete circuit that consists only of one Black Box with n inputs $\mathrm{req}_0 \ldots \mathrm{req}_{n-1}$, n outputs $\mathrm{ack}_0 \ldots \mathrm{ack}_{n-1}$ and a bounded memory of size m. If exact symbolic model checking for this circuit and the CTL formula $\varphi^n = \varphi_1^n \wedge \varphi_2^n \wedge \varphi_3^n$ states that this problem is realizable, then there is an implementation of the Black Box such that φ^n is satisfied.

Considering an arbiter for 2 clients, the implementation given in [4] has 4 flip-flops. However, our model checker was able to prove that for bounded memory size $m = 1$, there is an implementation of the Black Box satisfying φ^2 (but there is no memoryless implementation with $m = 0$). This result was achieved in 0.06 seconds with a peak live BDD node count of 667.

For 3 clients, the implementation shown in [4] has 6 flip-flops. Whereas we were able to show that 1 flip-flop is not sufficient (φ^3 'not realizable' with 1 flip-flop, shown in 0.39 seconds with a peak live BDD node count of 3162), we could prove that there is a realization with bounded memory size of 2. The proof was completed within 409.3 minutes with a peak live BDD node count of 13556734.

As an interesting side effect, if realizability can be shown, it is also possible to extract implementations realizing the property from the result of our model checking run: Having a closer look at the realizability check given by formula (11) of Sect. V (see page 11) one can see that every satisfying assignment to the $\vec{\mathbf{Z}}$ variables in $\forall \vec{x}\big(\chi_{Sat(\varphi^n)}|_{\vec{q}=\vec{q}^0}\big)$ represents a Black Box implementation satisfying the property.

In our experiments we obtained the result that for the arbiter with 2 clients, there is a total of 16777216 boolean functions the Black Box can be replaced with, whereof 288 substitutions satisfy φ^2. In the case of 3 clients, $1.1857 \cdot 10^{23}$ out of $1.4615 \cdot 10^{48}$ possible substitutions satisfy φ^3. The BDD that represented all substitutions satisfying φ^3 had a size of 1134840 nodes.

For the case of 3 clients we extracted one possible implementation by the following method: First we identified a shortest path from the 1-terminal to the root in the BDD representing $\forall \vec{x}\big(\chi_{Sat(\varphi^3)}|_{\vec{q}=\vec{q}^0}\big)$. The corresponding assignment to the $\vec{\mathbf{Z}}$ variables can be interpreted as the entries of a function table for the Black Box, thus giving an implementation. Variables with no assignment can be seen as don't-cares in the function table. Based on this, we used SIS [30] to obtain a minimized circuit. Interestingly, the resulting circuit even holds additional useful properties not required by φ^3: Every time a request is made, at least one of them is asserted. Additionally, all requests are asserted at the latest of two steps in the future (if the request is persistent).

The method presented in Sect. V is able to prove or disprove realizability or validity of properties for incomplete designs under the assumption that an upper bound on the amount of memory inside the Black Boxes is given. The method is exact also taking into account that the Black Boxes may have only restricted access to information present in the system, which is reflected by the fact that only a subset of the signals in the circuit is defined as the inputs of the Black Box. Whereas the method is able to provide interesting results as shown in this section, BDD sizes in our experiments also indicate that the exact method will be applicable to benchmarks of moderate size only. This again gives us a motivation for considering *approximate* methods for solving realizability and validity questions.

VII. Conclusions

We introduced a method that is able to use different methods for modeling unknowns at the outputs of Black Boxes within a single model checking run. This allows us to handle less relevant (in terms of the CTL formula) Black Boxes with larger approximation and thus faster, without necessarily losing important information only more exact methods can provide.

Experimental results using our implementation proved that the need for computational resources (both memory and time) could be substantially decreased by masking complex parts of the design and by using model checking for the resulting incomplete design. The increase of efficiency was obtained while still providing sound and useful results (even if the Black Boxes lie inside the cone of influence for the considered CTL formula).

Moreover, we presented a concept for exact symbolic model checking of incomplete designs containing several Black Boxes with bounded memory. This method is based on a reduction of the problem to a conventional model checking problem by applying transformations to the incomplete design at hand.

References

[1] 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.

[2] R. Bryant, "Graph - based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[3] 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.

[4] K. McMillan, *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[5] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: A New Symbolic Model Verifier," in *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, N. Halbwachs and D. Peled, Eds. Trento, Italy: Springer Verlag, July 1999, pp. 495–499. [Online]. Available: http://citeseer.ist.psu.edu/cimatti99nusmv.html

[6] The VIS Group, "VIS: A system for verification and synthesis," in *Computer Aided Verification*, ser. LNCS, vol. 1102. Springer Verlag, 1996, pp. 428–432.

[7] K. L. McMillan, *The SMV language*, Cadence Berkeley Labs, 1999.

[8] K. McMillan, *The SMV system - for SMV version 2.5.4*, Carnegie Mellon University, Nov. 2000.

[9] T. Villa, G. Swamy, and T. Shiple, *VIS User's Manual*, Electronics Research Laboratory, University of Colorado at Boulder, 1996.

[10] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *Design Automation Conf.*, 2001, pp. 238–243.

[11] T. Nopper and C. Scholl, "Approximate symbolic model checking for incomplete designs," in *Formal Methods in Computer-Aided Design*, ser. LNCS, A. J. Hu and A. K. Martin, Eds., vol. 3312. Austin, Texas: Springer Verlag, Nov 2004, pp. 290–305.

[12] A. Pnueli and R. Rosner, "Distributed systems are hard to synthesize," in *31th IEEE Symp. Found. of Comp. Science*, 1990, pp. 746–757.

[13] M. Huth, R. Jagadeesan, and D. Schmidt, "Modal transition systems: A foundation for three-valued program analysis," in *Proceedings of European Symposium on Programming*, D. Sands, Ed., vol. 2028. Springer, April 2001, pp. 155+.

[14] J. Burch and D. Dill, "Automatic verification of microprocessor control," in *Computer Aided Verification*, ser. LNCS, vol. 818. Springer Verlag, 1994, pp. 68–80.

[15] K. Sajid, A. Goel, H. Zhou, A. Aziz, and V. Singhal, "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions," in *Int'l. Conf. on CAV*, ser. LNCS, vol. 1447. Springer Verlag, 1998, pp. 244–255.

[16] S. Berezin, A. Biere, E. Clarke, and Y. Zhu, "Combining Symbolic Model Checking with Uninterpreted Functions for Out-Of-Order Processor Verification," in *Formal Methods in CAD*, 1998, pp. 369–386.

[17] R. Bryant, S. German, and M. Velev, "Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic," *ACM Trans. on Computational Logic*, vol. 2, no. 1, pp. 1–41, 2001.

[18] N. Yevtushenko, T. Villa, R. K. Brayton, A. Petrenko, and A. L. Sangiovanni-Vincentelli, "Solution of parallel language equations for logic synthesis," in *International Conference on Computer Aided Design*. IEEE Press, 2001, pp. 103–110.

[19] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

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

[21] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. Hsiao, "Testing, Verification, and Diagnosis in the Presence of Unknowns," in *VLSI Test Symp.*, 2000, pp. 263–269.

[22] A. Mishchenko and R. K. Brayton, "A theory of nondeterministic networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 977– 999, 2006.

[23] R. Bryant, "Symbolic Boolean manipulation with ordered binary decision diagrams," *ACM, Comp. Surveys*, vol. 24, pp. 293–318, 1992.

[24] T. Filkorn, "Functional extension of symbolic model checking," in *CAV '91: Proceedings of the 3rd International Workshop on Computer Aided Verification*. Springer, 1992, pp. 225–232.

[25] P. Williams, A. Biere, E. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking," in *Computer Aided Verification*, ser. LNCS, vol. 1855. Springer Verlag, 2000, pp. 124–138.

[26] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems II*. Springer, 1995, pp. 1–20.

[27] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[28] H. Higuchi and F. Somenzi, "Lazy group sifting for efficient symbolic state traversal of FSMs," in *Int'l Conf. on CAD*, 1999, pp. 45–49.

[29] R. Hojati, S. Krishnan, and R. Brayton, "Early quantification and partitioned transition relations," in *ICCD*, 1996, pp. 12–19.

[30] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," University of Berkeley, Tech. Rep., 1992.