Paper submitted to IEEE Trans. Computers.
Final paper appeared in IEEE Trans. Computers, vol. 62, no. 6, 2013, pp. 1234-1254.
http://doi.ieeecomputersociety.org/10.1109/TC.2012.53

Symbolic Model Checking for Incomplete Designs With Flexible Modeling of Unknowns

Tobias Nopper and Christoph Scholl

Abstract—We consider the problem of checking whether an incomplete design (i.e., a design containing 'unknown parts', 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. There are many applications of property checking for incomplete designs, such as early verification checks for unfinished designs, error localization in faulty designs and the abstraction of complex parts of a design in order to simplify the property checking task.

To process incomplete designs we present an approximate, yet sound algorithm. The algorithm is flexible in the sense that for every Black Box a different approximation method can be chosen. This permits us to handle less relevant Black Boxes (in terms of the property) 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 present a concept to decide *exactly* whether Black Boxes with bounded memory can be implemented so that they satisfy a given property. This question is reduced to conventional symbolic model checking.

The effectiveness and feasibility of the methods is demonstrated by a series of experimental results.

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

1 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 designs, Clarke, Emerson, and Sistla presented model checking for the temporal logic CTL [3]. Burch et al. improved the technique by using symbolic methods based on binary decision diagrams [4] for both state set representation and state traversal in [5], [6].

In this paper we consider how to perform model checking of *incomplete* designs, i.e., designs which contain unknown parts, combined into so-called Black Boxes. In doing so, we 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').

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

- *Parts of the article have been presented at DAC 2001 [1] and FMCAD 2004 [2].*

There are three major benefits symbolic model checking for incomplete designs 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 design.

In principle, the realizability problem could be solved *exactly* by synthesis approaches such as [7], [8]. Of course, a property for an incomplete design is realizable, if a complete design can be synthesized from the property and the parts of the design which are already known. However, due to complexity reasons, we are mainly interested in *approximate* solutions to the realizability and the validity problem. Whereas an *exact* solution to the realizability problem for incomplete designs with several Black Boxes (potentially containing an unrestricted amount of memory) is even undecidable in general [9], we use symbolic methods providing approximate answers: Our algorithm does not give a definite answer in every case, but it is guaranteed to be sound in the sense that it never gives an incorrect answer; it provides proofs of validity and disproofs of realizability for arbitrary CTL formulas (unlike approaches using 'non-deterministic signals' implemented in SMV [6] and VIS [10] which are only sound for certain subclasses of CTL).

Our method is based on symbolic representations of incomplete designs. 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. Approximate yet sound answers for realizability and validity are computed based on these sets.

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 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.

For an experimental evaluation we considered pipelined ALUs with varying bit widths. The results show that our approximate methods are able to provide proofs for large designs with bit widths up to 64, whereas standard model checking succeeded only for much smaller benchmarks. Moreover, the results show that the flexibility of choosing approximations with different accuracy for different Black Box outputs is essential for the success of our method. These observations were also confirmed by an additional case study from the railway domain.

Although the main focus of our paper lies on approximate solutions to the realizability and validity problems — in order to make our presentation more complete — we also present a concept how to provide an *exact* solution to a restricted problem by means of a *conventional symbolic model checker*: We assume an upper bound to the number of internal states of the Black Boxes and symbolically compute the exact set of Black Box replacements for which a property is satisfied, i.e., we give exact answers to both the realizability and the validity question. In contrast to controller synthesis approaches such as [7], [8], we do not assume that Black Boxes have unlimited access to all signals in the design, but we take into account that they are only able to read the input signals connected to them. In Sect. 7 we applied this concept to a design where we checked the realizability of an arbiter which was specified by CTL formulas.

Our approach shares ideas with 3-valued model checking introduced in the context of software model checking (e.g. [11]–[13]); it extends these ideas, improves and adapts them making use of characteristics of modular hardware designs and it provides an efficient implementation based on symbolic methods. Compared with methods from hardware verification such as Symbolic Trajectory Evaluation (STE) [14] or verification using Uninterpreted Functions (UIFs) [15], our method supports full CTL and allows Black Boxes for *sequential* designs. Our approach (in its aspect of abstraction by Black Boxes) is also related to localization reduction [16]. Bounded model checking approaches with localization reduction (e.g. [17], [18]) make use of efficient SAT solvers and are restricted to safety properties. If the property allows a non-trivial amount of abstraction of the full model, our BDD based symbolic method proves to be competitive for large designs (and even for saftey properties) due to property specific abstractions of different strengths. The method is based on user knowledge about the design and on user assumptions about the importance of certain parts of the design for the property at hand. Especially if the interfaces of some Black Boxes are wide (i.e. contain many signals), by our flexible modeling of unknowns we are not only able to abstract complex implementations of Black Boxes, but also to reduce the number of variables for interface signals (which is an additional source of complexity). A more detailed discussion of the relationship between our work and other approaches from the literature can be found in Sect. 8.

The paper is structured as follows: After giving a brief review of sequential designs and symbolic model checking in Sect. 2, we define incomplete designs and the set of their completions in Sect. 3. Sect. 4 introduces our method to perform symbolic simulation for incomplete designs and in Sect. 5, we present a new algorithm capable of performing sound and approximate symbolic model checking for incomplete designs. In Sect. 6, we 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 Black Box. We give a series of experimental results demonstrating the effectiveness and feasibility of the methods in Sect. 7. Finally, Sect. 8 provides a detailed discussion of related work and Sect. 9 concludes the paper.

2 Preliminaries

Before we introduce symbolic model checking for incomplete designs we give a brief review of symbolic model checking for complete designs [5]. Symbolic model checking is applied to Kripke structures (which may be derived from sequential designs) on the one hand and to a formula of a temporal logic (in our case CTL ('Computation Tree Logic')) on the other hand.

2.1 Sequential Designs and Kripke Structures

(Complete) sequential designs consist of nodes which are connected by signals.[1] The nodes represent primary inputs, primary outputs, memory elements (flip-flops) storing single bits, or logic gates implementing Boolean functions. In following we denote the list of primary input nodes by $\vec{X} = (X_1, X_2, \ldots, X_{|\vec{X}|})$, the list of primary output nodes by $\vec{Y} = (Y_1, Y_2, \ldots, Y_{|\vec{Y}|})$, and the list of flip-flops by $\vec{Q} = (Q_1, Q_2, \ldots, Q_{|\vec{Q}|})$. $\vec{q}^{\,0} \in \mathbb{B}^{|\vec{Q}|}$ gives the initial values (initial state) of the flip-flops. Fig. 1 a) shows an example for a sequential design with one primary input, three primary outputs, two flip-flops initialized to 0, and three gates implementing Boolean functions nor_2, and_2, xor_2, respectively.

Each output of a node in the sequential design computes a Boolean function $f\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}$. Each primary input X_j computes a Boolean function x_j (which – strictly speaking – is the projection function mapping $(x_1, \ldots, x_{|\vec{X}|}, q_1, \ldots, q_{|\vec{Q}|}) \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|}$ to x_j) and each flip-flop Q_k computes q_k. The output functions of the logic gates are computed recursively according to their gate function. In Fig. 1 a) the corresponding Boolean functions of the nodes are shown as Boolean expressions.

In symbolic model checking, BDD based representations of these Boolean functions are computed by *symbolic simulation* [19]. For this, the primary inputs and the outputs of the flip-flops are associated with unique BDD variables and BDDs for the functions computed by the gates of the design are built in topological order using BDD operations.

The input functions δ_i of the flip-flops Q_i compute the next state of the flip flops and the functions λ_j corresponding to the primary outputs compute the current output values (based on the current state and the current input). Altogether a sequential design defines a *transition function* $\vec{\delta}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Q}|}$ and an *output function* $\vec{\lambda}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Y}|}$. Again, see Fig. 1 a) for an example.

Now we can define the Kripke structure of a complete design D. The states of the Kripke structure are defined as a combination of states and inputs of D. The transition relation R of the Kripke structure connects states according to the transition function $\vec{\delta}$ of D. The labeling function L labels states with the information which inputs (represented by atomic propositions x_i) and outputs (represented by atomic propositions y_i) are 1 in these states.

Definition 1 (Kripke Structure of a Complete Design)**.** *Let D be a sequential design with transition function $\vec{\delta}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Q}|}$, output function $\vec{\lambda}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Y}|}$, and initial state $\vec{q}^{\,0}$. A Kripke structure for D is $struct(D) := (S, R, L)$ where*

- $S := \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|}$, $R \subseteq S \times S$, $L\colon S \to V$,
- $V := \{x_1, \ldots, x_{|\vec{X}|}\} \cup \{y_1, \ldots, y_{|\vec{Y}|}\}$,
- $R := \{((\vec{q}, \vec{x}), (\vec{q}', \vec{x}')) \mid \vec{q}, \vec{q}' \in \mathbb{B}^{|\vec{Q}|}, \vec{x}, \vec{x}' \in \mathbb{B}^{|\vec{X}|}, \vec{\delta}(\vec{q}, \vec{x}) = \vec{q}'\}$,
- *and* $L((\vec{q}, \vec{\epsilon})) := \{x_i \mid \epsilon_i = 1\} \cup \{y_i \mid \lambda_i(\vec{q}, \vec{\epsilon}) = 1\}$.

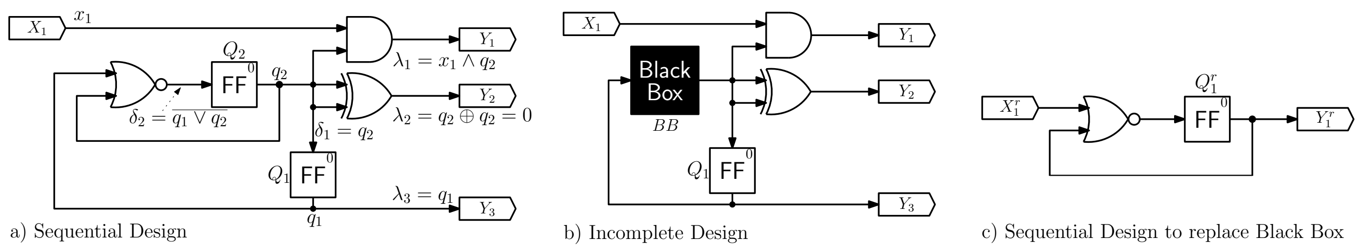
All states $(\vec{q}^{\,0}, \vec{x})$ *with* $\vec{x} \in \mathbb{B}^{|\vec{X}|}$ *are called* initial states *of* $struct(D)$.

2.2 Symbolic Model Checking for Complete Designs

Model checking decides whether a given design fulfills its specification given as a formula of a temporal logic. In this paper we consider the widespread branching time logic CTL (Computation Tree Logic) [3], [6]. CTL formulas specify properties of states of Kripke structures. The semantics of CTL formulas can be defined recursively based on their structure:

Definition 2 (Semantics of CTL)**.** *As usual we write $struct(D), s \models \varphi$ if the CTL formula φ is satisfied in state $s = (\vec{q}, \vec{x}) \in S$ of $struct(D)$. If it is clear from the context which Kripke structure is*

1. For a formal definition see Appendix A.



a) Sequential Design b) Incomplete Design c) Sequential Design to replace Black Box

Fig. 1. Complete and incomplete sequential designs

used, we simply write $s \models \varphi$ instead of $struct(D), s \models \varphi$. "$\models$" is defined as follows:

$$
\begin{array}{lll}
s \models \varphi;\ \varphi \in V & \Longleftrightarrow & \varphi \in L(s) \\
s \models \neg\varphi & \Longleftrightarrow & s \not\models \varphi \\
s \models (\varphi_1 \vee \varphi_2) & \Longleftrightarrow & s \models \varphi_1 \textit{ or } s \models \varphi_2 \\
s \models EX\varphi & \Longleftrightarrow & \exists s' \in S\colon R(s,s') \textit{ and } s' \models \varphi \\
s \models EG\varphi & \Longleftrightarrow & \textit{there is a path } (s_0, s_1, s_2, \dots) \textit{ with} \\
 & & s = s_0 \textit{ and } \forall i \geq 0\colon (s_i, s_{i+1}) \in R \textit{ and } s_i \models \varphi \\
s \models E\varphi_1 U \varphi_2 & \Longleftrightarrow & \textit{there is a path } (s_0, s_1, s_2, \dots) \textit{ with} \\
 & & s = s_0 \textit{ and } \forall i \geq 0\colon (s_i, s_{i+1}) \in R \textit{ and there is} \\
 & & j \textit{ so that } s_j \models \varphi_2 \textit{ and } \forall 0 \leq i < j\colon s_i \models \varphi_1
\end{array}
$$

The remaining CTL operations $\wedge$, EF, AX, AU, AG, and AF can be expressed by using $\neg$, $\vee$, EX, EU, and EG [6].

In *symbolic* model checking, sets of states are represented by characteristic functions (which are in turn represented by BDDs). Before we define symbolic model checking, we need the following three definitions:

Definition 3 (Cofactor)**.** *For a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ the cofactor (or partial evaluation) wrt. $\vec{y} = \vec{\epsilon}$ ($\vec{\epsilon} \in \mathbb{B}^{|\vec{y}|}$) is defined as the Boolean function $f|_{\vec{y}=\vec{\epsilon}} : \mathbb{B}^n \to \mathbb{B}$ with $f|_{\vec{y}=\vec{\epsilon}}(\vec{x}, \vec{y}, \vec{z}) = f(\vec{x}, \vec{\epsilon}, \vec{z})$ for all $(\vec{x}, \vec{y}, \vec{z}) \in \mathbb{B}^n$.*

Definition 4. *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a Boolean function. $\exists \vec{y} f$ is defined as the Boolean function $\bigvee_{\vec{\epsilon} \in \mathbb{B}^{|\vec{y}|}} f|_{\vec{y}=\vec{\epsilon}}$, $\forall \vec{y} f$ as $\bigwedge_{\vec{\epsilon} \in \mathbb{B}^{|\vec{y}|}} f|_{\vec{y}=\vec{\epsilon}}$.*

Definition 5 (Compose)**.** *Let $f, g : \mathbb{B}^n \to \mathbb{B}$ be Boolean functions. The composition of x_i by g in f is defined as the Boolean function $f|_{x_i \leftarrow g} : \mathbb{B}^n \to \mathbb{B}$ with $f|_{x_i \leftarrow g}(x_1, \dots, x_i, \dots, x_n) := f(x_1, \dots g(x_1, \dots, x_n), \dots, x_n) = (\overline{g} \cdot f|_{x_i=0} + g \cdot f|_{x_i=1})(x_1, \dots, x_n)$ for all $(x_1, \dots, x_n) \in \mathbb{B}^n$.*

Composition (see e.g. [4]) can be naturally generalized to vectors of variables and functions. In the definition of symbolic model checking only the special case of renaming variables is needed.

Let $Sat(\varphi)$ be the set of states of $struct(D)$ 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 $\chi_R(\vec{q}, \vec{x}, \vec{q}') := \prod_{i=1}^{|\vec{q}|} \left(\delta_i(\vec{q}, \vec{x}) \equiv q_i'\right)$ of the transition relation R:

$$
\begin{array}{ll}
\chi_{Sat(x_i)}(\vec{q}, \vec{x}) & := x_i \\
\chi_{Sat(y_i)}(\vec{q}, \vec{x}) & := \lambda_i(\vec{q}, \vec{x}) \\
\chi_{Sat(\neg\varphi)}(\vec{q}, \vec{x}) & := \overline{\chi_{Sat(\varphi)}}(\vec{q}, \vec{x}) \\
\chi_{Sat((\varphi_1 \vee \varphi_2))}(\vec{q}, \vec{x}) & := \chi_{Sat(\varphi_1)}(\vec{q}, \vec{x}) \vee \chi_{Sat(\varphi_2)}(\vec{q}, \vec{x}) \\
\chi_{Sat(EX\varphi)}(\vec{q}, \vec{x}) & := \chi_{EX}(\chi_{Sat(\varphi)})(\vec{q}, \vec{x}) \\
\chi_{Sat(EG\varphi)}(\vec{q}, \vec{x}) & := \chi_{EG}(\chi_{Sat(\varphi)})(\vec{q}, \vec{x}) \\
\chi_{Sat(E\varphi_1 U \varphi_2)}(\vec{q}, \vec{x}) & := \chi_{EU}(\chi_{Sat(\varphi_1)}, \chi_{Sat(\varphi_2)})(\vec{q}, \vec{x}) \\
\text{with } \chi_{EX}(\chi_N)(\vec{q}, \vec{x}) & := \exists \vec{q}' \exists \vec{x}' \Big(\chi_R(\vec{q}, \vec{x}, \vec{q}') \cdot \big(\chi_N|_{\substack{\vec{q} \leftarrow \vec{q}' \\ \vec{x} \leftarrow \vec{x}'}}\big)(\vec{q}', \vec{x}')\Big)
\end{array}
$$

χ_{EG} and χ_{EU} can be evaluated by the fixed point iteration algorithms shown in Figs. 2 and 3.

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

Fig. 2. Fixed point iteration for EG

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

Fig. 3. Fixed point iteration for EU

A complete sequential design satisfies a formula φ iff φ is satisfied in all initial states of the corresponding Kripke structure $struct(D)$:

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

3 Incomplete Designs

Incomplete sequential designs may contain additional "Black Box" (BB) nodes which represent parts of the design with unknown (sequential) behavior, see Fig. 1 b) for an example with one input, three outputs, one flip-flop, two gates implementing the Boolean and_2 resp. the Boolean xor_2 function and one Black Box.

A Black Box BB in an incomplete design D can be replaced by any sequential design D^r (without Black Boxes and with an arbitrary number of flip-flops), as long as D^r has the same number of inputs and outputs as the Black Box. The inputs and outputs of D^r are then connected to the inputs and outputs of the former Black Box BB in D; the result of this substitution is another (possibly incomplete) sequential design D^c. E.g. the sequential design in Fig. 1 a) results from the incomplete design in Fig. 1 b) by replacing the Black Box with the sequential design in Fig. 1 c).[2]

If the incomplete design D resulted from a complete design by abstractions replacing subcircuits by Black Boxes, then replacing Black Boxes again by their concrete counterparts corresponds to the well-known notion of *abstraction refinement* [16].

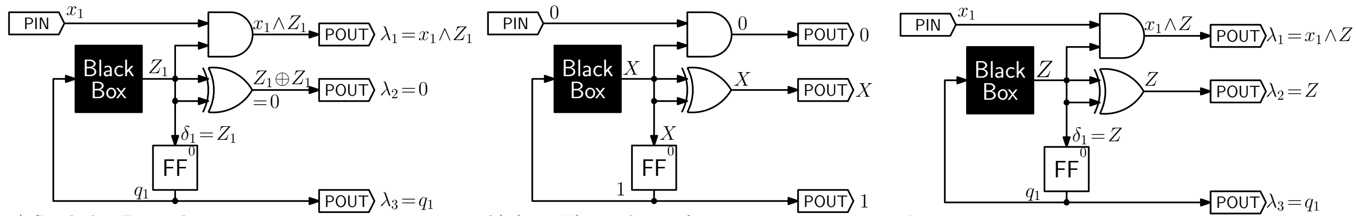
Definition 6 (Completion of an Incomplete Design)**.** *A sequential design D^c that was constructed from an incomplete design D by replacing* all *Black Boxes by sequential designs is called a* completion *of D. $\mathcal{C}(D)$ is the set of all possible completions of D.*

The two main questions we address in this paper are the realizability and the validity question:

Definition 7 (Realizability and Validity)**.** *Given an incomplete design D and a CTL formula φ:*

1) *If there is a completion $D^c \in \mathcal{C}(D)$ of D which satisfies φ, then the property φ is called* realizable *for D.*
2) *If all possible completions $D^c \in \mathcal{C}(D)$ of D satisfy φ, then the property φ is called* valid *for D.*

2. For formal definitions see Appendix A.



a) Symbolic Z_i-simulation b) $(0,1,X)$-simulation for $x_1 = 0, q_1 = 1$ c) Symbolic Z-simulation

Fig. 4. Different methods to analyze an incomplete design

4 Symbolic Simulation for Incomplete Designs

For symbolic CTL model checking of complete designs, symbolic representations of the output functions $\vec{\lambda}$ and the transition functions $\vec{\delta}$ are needed first. Of course, we cannot compute $\vec{\lambda}$ and $\vec{\delta}$ for *incomplete* designs due to the unknown Black Boxes. However, in order to define an approximate model checking method for *incomplete* designs in Sect. 5, we compute 'approximate representations' of output functions $\vec{\lambda}$ and transition functions $\vec{\delta}$ which contain information on the potential effect of the Black Boxes.

Symbolic Z_i-simulation

For that purpose we first consider *symbolic Z_i-simulation* which replaces the Black Box outputs by free input variables and in that way evaluates the effect that Black Box outputs have on $\vec{\lambda}$ and $\vec{\delta}$. Apart from handling the Black Box outputs as additional inputs, symbolic Z_i-simulation works exactly as conventional symbolic simulation [19]. (Replacing signals by free variables is not a new idea, but has been used for a long time, e.g. for localization abstraction [16].)

Definition 8 (Symbolic Z_i-Simulation)**.** *Let D be an incomplete design over the library $STD = \{and_2, or_2, not\}$.*[3] *Let zvar be a function mapping* distinct *Boolean variables Z_i to the outputs of the Black Boxes. Moreover, we use Boolean variables $x_1, \ldots, x_{|\vec{X}|}$ for the primary inputs and Boolean variables $q_1, \ldots, q_{|\vec{Q}|}$ for the flip-flop outputs. The Boolean function $f_{Z_i}(m, j)$ for the j-th output of a node m is defined as follows:*

(1) *If m is a primary input X_i, then $f_{Z_i}(m, 1) = x_i$.*
(2) *If m is a flip-flop Q_k, then $f_{Z_i}(m, 1) = q_k$.*
(3) *If m is a Black Box, then $f_{Z_i}(m, j) = zvar(m, j)$.*
(4) *If m is a not-gate whose predecessor is output k of node p, then $f_{Z_i}(m, 1) = \overline{f_{Z_i}(p, k)}$.*
(5) *If m is an and_2-gate (or_2-gate) whose predecessors are output k_1 of p_1 and output k_2 of p_2, then $f_{Z_i}(m, 1) = f_{Z_i}(p_1, k_1) \wedge f_{Z_i}(p_2, k_2)$ ($f_{Z_i}(p_1, k_1) \vee f_{Z_i}(p_2, k_2)$).*

Fig. 4 a) shows an example for symbolic Z_i-simulation of the incomplete design from Fig. 1 b).

The following lemma shows how the result of symbolic Z_i-simulation can be interpreted regarding the Black Boxes:

Lemma 1. *Let D be an incomplete design and let $D^c \in \mathcal{C}(D)$ be an arbitrary completion of D. For the j-th output of gate m in D let $f_{Z_i}(m, j)$ be the Boolean function over variables $(\vec{q}, \vec{x}, \vec{Z})$ computed by symbolic Z_i-simulation of D. Furthermore, let $f(m, j)$ be the Boolean function over variables $((\vec{q}, \vec{q}^{\,r}), \vec{x})$ computed by (conventional) symbolic simulation of D^c ($\vec{q}^{\,r}$ are variables introduced by substitutions of Black Boxes by sequential designs). Then the following holds for constants $\alpha \in \mathbb{B}$ and $\vec{\beta} \in \mathbb{B}^{|\vec{q}|}$, $\vec{\gamma} \in \mathbb{B}^{|\vec{x}|}$:*

$$f_{Z_i}(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} = \alpha \Rightarrow f(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} = \alpha.$$

Proof:
The proof simply follows from the fact that $f_{Z_i}(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} = \alpha \in \mathbb{B}$ implies that $f_{Z_i}(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}}$ does not depend on the outputs of the Black Boxes. Therefore $f(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} = \alpha$ for an arbitrary substitution of the Black Boxes by sequential designs. □

3. W.l.o.g. we restrict the library in the following to STD, since all types of gates can be expressed using 2-input and_2 gates, or_2 gates and not gates.

Symbolic Z-simulation

For analyzing combinational circuits with Black Boxes in [1] we introduced symbolic Z-simulation. Compared to symbolic Z_i-simulation, this method is usually less expensive in terms of run time and memory consumption, but it is also less accurate as measured by the amount of information which can be extracted from the results.

Symbolic Z-simulation is motivated by the well-known $(0, 1, X)$-simulation [14], [20], [21]. The value X represents unknown values which come from the unknown functionality of the Black Boxes in our context. If some input values of a gate are set to X during $(0, 1, X)$-simulation, the output value 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. 4 b) shows a (conventional) $(0, 1, X)$-simulation for the incomplete design shown in Fig. 1 b) (with the input set to 0 and the flip-flop state set to 1).

For symbolic Z-simulation, the *symbolic* version of $(0, 1, X)$-simulation, a new variable Z is introduced, which is used to model unknown values at the outputs of Black Boxes. Now, for each output of a node in the incomplete design, the output function is obtained by using a slightly modified version of symbolic simulation:

Definition 9 (Symbolic Z-Simulation)**.** *Let D be an incomplete design over $STD = \{and_2,\ or_2, not\}$. Let Z be a new variable different from $x_1, \ldots, x_{|\vec{X}|}, q_1, \ldots, q_{|\vec{Q}|}$. The Boolean function $f_Z(m, j)$ for the j-th output of a node m is defined as in Def. 8 with (3) and (4) replaced by*

(3') *If m is a Black Box, then $f_Z(m, j) = Z$.*
(4') *If m is a not-gate whose predecessor is output k of node p, then $f_Z(m, 1) = \overline{f_Z(p, k)}|_{Z \leftarrow \overline{Z}}$.*

The main difference to conventional symbolic simulation is the evaluation of not-gates: The not operation on the function for the predecessor gate is followed by a $compose$ operation (see Def. 5) which composes $\overline{Z}$ for Z (written as $\overline{f_Z(p, k)}|_{Z \leftarrow \overline{Z}}$). Fig. 5 (a) shows a first example of Z-simulation for a design with two Black Boxes. If the compose operation after processing the not-gate would be omitted, then its output would compute $\overline{Z}$, leading to the result 0 at the output of the and_2-gate, i.e. we would then lose the information that the output value is always unknown (modeled by Z) due to the Black Boxes.

Symbolic Z-simulation has the following property:

Lemma 2. *Given an incomplete design D and a Boolean function $f_Z(m, j)$ computed by symbolic Z-simulation for the j-th output of node m. For all $\vec{\beta} \in \mathbb{B}^{|\vec{q}|}$, $\vec{\gamma} \in \mathbb{B}^{|\vec{x}|}$: $f_Z(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} \in \{0, 1, Z\}$.*

The lemma can be proved by induction on the structure of D. The proof is given in Appendix B. If $f_Z(m, j)|_{\substack{\vec{q}=\vec{\beta} \\ \vec{x}=\vec{\gamma}}} = Z$, then the function value for input $(\vec{\beta}, \vec{\gamma})$ is unknown due to the Black Boxes.

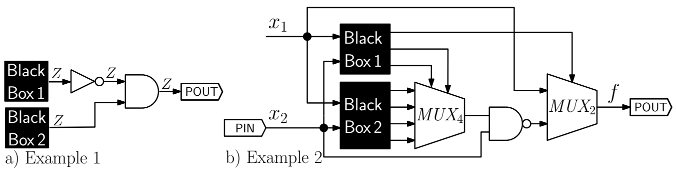


Fig. 5. Two incomplete designs

Fig. 4 c) shows another example of symbolic Z-simulation. Note that — in contrast to symbolic Z_i-simulation in Fig. 4 a) — the second output cannot be proved to be constant 0. Since Z-simulation cannot distinguish between unknown values at different Black Box outputs, some information is lost. According to the following lemma, Z_i-simulation is always at least as accurate as Z-simulation:

Lemma 3. *Let D be an incomplete design, $f_Z(m,j)$ the function computed by symbolic Z-simulation for the j-th output of m and $f_{Z_i}(m,j)$ the corresponding function computed by symbolic Z_i-simulation. Then the following holds for constants $\alpha \in \mathbb{B}$ and $\vec{\beta} \in \mathbb{B}^{|\vec{q}|}$, $\vec{\gamma} \in \mathbb{B}^{|\vec{x}|}$: If $f_Z(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha$, then $f_{Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha$.*

Again, the lemma is proved by induction on the structure of D; the proof can be found in Appendix C.

Symbolic Z/Z_i-simulation

Finally, we provide a mixed (symbolic) Z/Z_i-simulation in order to give the user more flexibility in controlling the trade-off between higher efficiency of Z-simulation and higher accuracy of Z_i-simulation. Here, some Black Box outputs are represented by variable Z as in symbolic Z-simulation and some Black Box outputs by distinct variables Z_i as in symbolic Z_i-simulation. Basically, symbolic Z/Z_i-simulation considers the Z_i-modeled Black Box outputs as additional inputs and then performs symbolic Z-simulation (always replacing Z by $\overline{Z}$ when processing not gates):

Definition 10 (Symbolic Z/Z_i-Simulation)**.** *Let D be an incomplete design over STD. Let $\vec{Z}_l = (Z_1,\dots,Z_n)$ be a vector of new variables and Z be a new variable, all different from $x_1,\dots,x_{|\vec{X}|}, q_1,\dots,q_{|\vec{Q}|}$. Let zvar be a function mapping* distinct *Boolean variables Z_i* or *the variable Z to the outputs of the Black Boxes. The Boolean function $f_{Z/Z_i}(m,j)$ for the j-th output of a node m is defined as in Def. 8 with (4) replaced by*

(4") *If m is a not-gate whose predecessor is output k of node p, then $f_{Z/Z_i}(m,1) = \overline{f_{Z/Z_i}(p,k)}|_{Z \leftarrow \overline{Z}}$.*

Example. Figure 5 (b) shows an example comparing Z-, Z_i- and combined Z/Z_i-simulation. If this design is simulated by using symbolic Z-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 design 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 $(x_1, x_2, Z_1, \dots, Z_7)$, 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 $Z_1, \dots Z_7$ are assigned top down to the Black Box outputs appearing in Fig. 5 (b).

When using symbolic Z/Z_i-simulation 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 $f_{Z/Z_i} = \overline{Z}_1 x_1 + Z_1 \cdot (\overline{x}_2 + Z)$.

Thus, Z/Z_i-simulation 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 Z-simulation. To give an example, for $x_1 = 1$ and $x_2 = 0$, the output can be proven to be 1 using Z/Z_i-simulation, while it is not possible to obtain this information from symbolic Z-simulation.

In general, Z/Z_i-simulation is at most as exact as symbolic Z_i-simulation, but at least as exact as symbolic Z-simulation. Moreover, if for a node function in an incomplete design computed by Z-, Z/Z_i- or Z_i-simulation a cofactor wrt. input and state variables is constant, then the cofactor of the corresponding node function always evaluates to the same constant, no matter how the Black Boxes are replaced by sequential designs. This is summarized by the following theorem:

Theorem 4. *Let D be an incomplete design, let $D^c \in \mathcal{C}(D)$ be an arbitrary completion of D, let $f_Z(m,j)$, $f_{Z_i}(m,j)$, $f_{Z/Z_i}(m,j)$ be the functions computed for the j-th output of node m in D by symbolic Z-simulation, Z_i-simulation, and Z/Z_i-simulation, respectively, and let $f(m,j)$ be the function computed for the j-th output of node m in D^c by symbolic simulation. For constants $\alpha \in \mathbb{B}$ and for all $\vec{\beta} \in \mathbb{B}^{|\vec{q}|}$, $\vec{\gamma} \in \mathbb{B}^{|\vec{x}|}$:*

$$f_Z(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha \overset{(1)}{\Rightarrow} f_{Z/Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha$$
$$\overset{(2)}{\Rightarrow} f_{Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha \overset{(3)}{\Rightarrow} f(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha.$$

Proof: Implication (1) is proved by induction on the structure of D exactly as in the proof of Lemma 3 (simply replace f_{Z_i} by f_{Z/Z_i} in the proof). Implication (2) follows from Lemma 3: Let $\vec{Z}_l$ be the vector of Z_i-variables also used in Z/Z_i-simulation. Since $f_{Z/Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha$, also $f_{Z/Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}\\ \vec{Z}_l=\vec{\epsilon}}} = \alpha$ for all $\vec{\epsilon} \in \mathbb{B}^{|\vec{Z}_l|}$. Considering the $\vec{Z}_l$-variables as primary inputs for the time being, we can apply Lemma 3 and have $f_{Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}\\ \vec{Z}_l=\vec{\epsilon}}} = \alpha$ for all $\vec{\epsilon} \in \mathbb{B}^{|\vec{Z}_l|}$. This implies $f_{Z_i}(m,j)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = \alpha$. Implication (3) directly follows from Lemma 1. □

5 Symbolic Model Checking for Incomplete Designs

5.1 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 design satisfies φ. The situation becomes more complex if we consider incomplete designs, since for each replacement of the Black Boxes we may have different state sets satisfying φ.

In contrast to conventional model checking we do not compute the set $Sat(\varphi)$, but we consider two sets $Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{exact}}(\varphi)$: The set $Sat_E^{\mathsf{exact}}(\varphi)$ is defined to contain all states, for which *there is* at least one completion so that φ is satisfied. In a similar manner, $Sat_A^{\mathsf{exact}}(\varphi)$ contains all states, for which φ is satisfied for *all* possible completions.

Definition 11. *Let D be an incomplete design and $\mathcal{C}(D)$ be the set of completions of D.*

$$Sat_E^{\mathsf{exact}}(\varphi) := \Big\{ (\vec{q},\vec{x}) \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \mid \exists D^c \in \mathcal{C}(D), \exists \vec{q}^{\,r} \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|} : struct(D^c), ((\vec{q},\vec{q}^{\,r}),\vec{x}) \models \varphi \Big\}$$
$$Sat_A^{\mathsf{exact}}(\varphi) := \Big\{ (\vec{q},\vec{x}) \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \mid \forall D^c \in \mathcal{C}(D), \forall \vec{q}^{\,r} \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|} : struct(D^c), ((\vec{q},\vec{q}^{\,r}),\vec{x}) \models \varphi \Big\}$$

($|\vec{Q}^c| - |\vec{Q}|$ is the number of flip-flops in D^c added to D by Black Box replacements.) We say that states $(\vec{q},\vec{x}) \in Sat_E^{\mathsf{exact}}(\varphi)$ 'possibly satisfy φ' and that states $(\vec{q},\vec{x}) \in Sat_A^{\mathsf{exact}}(\varphi)$ 'definitely satisfy φ'.

Of course, Def. 11 does not suggest a feasible algorithm for computing $Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{exact}}(\varphi)$, since the set of all possible completions for an incomplete design is not finite. (This motivates our approach to compute approximations of $Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{exact}}(\varphi)$

in Sect. 5.2.) Nevertheless, given $Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{exact}}(\varphi)$, it is easy to prove validity and to falsify realizability for the incomplete design:

Lemma 5. *A property φ is valid for an incomplete design D with initial state $\vec{q}^0$, if all states $(\vec{q}^0, \vec{x})$ with $\vec{x} \in \mathbb{B}^{|\vec{X}|}$ are included in $Sat_A^{\mathsf{exact}}(\varphi)$. A property φ is not realizable for D, if there is at least one such state $(\vec{q}^0, \vec{x})$ not belonging to $Sat_E^{\mathsf{exact}}(\varphi)$.*

Proof: Let $D^c \in \mathcal{C}(D)$ be an arbitrary completion of D and let $\vec{q}^{0r}$ be the initial states of the flip-flops introduced by the replacements of the Black Boxes. If $\forall \vec{x} \in \mathbb{B}^{|\vec{X}|}$: $(\vec{q}^0, \vec{x}) \in Sat_A^{\mathsf{exact}}(\varphi)$, then $\forall \vec{x} \in \mathbb{B}^{|\vec{X}|}$: $struct(D^c), ((\vec{q}^0, \vec{q}^{0r}), \vec{x}) \models \varphi$ (by Def. 11). This means that D^c satisfies φ. φ is valid, since we assumed D^c to be an arbitrary completion. If $(\vec{q}^0, \vec{x}) \notin Sat_E^{\mathsf{exact}}(\varphi)$, then for all completions $D^c \in \mathcal{C}(D)$ with initial states $\vec{q}^{0r}$ of the additional flip-flops φ is not satisfied in the initial state $((\vec{q}^0, \vec{q}^{0r}), \vec{x})$ of $struct(D^c)$. Thus, φ is not realizable for D. □

Just as Black Boxes in incomplete designs lead to states only *possibly* satisfying φ, there are also '*possible transitions*' between states in an incomplete design which may or may not exist in a completion of the design — depending on the replacement of the Black Boxes:

Definition 12. *Let D be an incomplete design and let $\mathcal{C}(D)$ be the set of completions of D. We define the incomplete design to have a* possible transition *between states $(\vec{q}, \vec{x}), (\vec{q}', \vec{x}') \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|}$, if there is a completion $D^c \in \mathcal{C}(D)$ for which there is a transition between $((\vec{q}, \vec{q}^r), \vec{x})$ and $((\vec{q}', \vec{q}'^r), \vec{x}')$ for some values $\vec{q}^r$, $\vec{q}'^r \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$. ($|\vec{Q}^c| - |\vec{Q}|$ is the number of flip-flops in D^c added to D by Black Box replacements.)*

Possible transitions are used later on in order to *compute* states that possibly or definitely satisfy a property φ.

5.2 Approximations

As mentioned above, for reasons of efficiency we compute *approximations* $Sat_E^{\mathsf{appr}}(\varphi)$ and $Sat_A^{\mathsf{appr}}(\varphi)$ for $Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{exact}}(\varphi)$, respectively – and similarly for the set of possible transitions. To be more precise, we compute over-approximations $Sat_E^{\mathsf{appr}}(\varphi) \supseteq Sat_E^{\mathsf{exact}}(\varphi)$ of $Sat_E^{\mathsf{exact}}(\varphi)$ and under-approximations $Sat_A^{\mathsf{appr}}(\varphi) \subseteq Sat_A^{\mathsf{exact}}(\varphi)$ of $Sat_A^{\mathsf{exact}}(\varphi)$.

Then Lemma 5 directly implies the following lemma:

Lemma 6. *Let $Sat_E^{\mathsf{appr}}(\varphi) \supseteq Sat_E^{\mathsf{exact}}(\varphi)$ and $Sat_A^{\mathsf{appr}}(\varphi) \subseteq Sat_A^{\mathsf{exact}}(\varphi)$. If all initial states $(\vec{q}^0, \vec{x})$ are included in $Sat_A^{\mathsf{appr}}(\varphi)$, then φ is valid. If there is an initial state $(\vec{q}^0, \vec{x})$ that is not included in $Sat_E^{\mathsf{appr}}(\varphi)$, then φ is not realizable.*

Approximations $Sat_E^{\mathsf{appr}}(\varphi)$ and $Sat_A^{\mathsf{appr}}(\varphi)$ are computed based on an approximate transition relation and on approximate output functions for the incomplete design D.

To take account of the unknown behavior of the Black Boxes in D we use the symbolic methods from Sect. 4: Let there be a number of Black Box outputs modeled by Z and some other Black Box outputs modeled by Z_i–variables from $\{Z_1, \ldots, Z_n\}$. Symbolic Z/Z_i-simulation computes 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 $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|}$ if $\lambda_i|_{\vec{q}=\vec{q}^{\mathsf{fix}}, \vec{x}=\vec{x}^{\mathsf{fix}}} = 1$. Here we include a state $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ into $Sat_A^{\mathsf{appr}}(y_i)$, if λ_i is 1 for a fixed value $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ assigned to $(\vec{q}, \vec{x})$ and *all* possible assignments to Z and $\vec{Z}_l$. We include $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ into $Sat_E^{\mathsf{appr}}(y_i)$, if λ_i is 1 for $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ assigned to $(\vec{q}, \vec{x})$ and *some* assignment to Z and $\vec{Z}_l$. Thus we define the characteristic functions of $Sat_A^{\mathsf{appr}}(y_i)$ and $Sat_E^{\mathsf{appr}}(y_i)$ as follows:

Definition 13.

$$\chi_{Sat_A^{\mathsf{appr}}(y_i)}(\vec{q}, \vec{x}) := \forall Z \forall \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l)\big) \tag{1}$$
$$\chi_{Sat_E^{\mathsf{appr}}(y_i)}(\vec{q}, \vec{x}) := \exists Z \exists \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l)\big) \tag{2}$$

Lemma 7. *For $Sat_A^{\mathsf{appr}}(y_i)$ and for $Sat_E^{\mathsf{appr}}(y_i)$ as defined in Def. 13:*

$$Sat_A^{\mathsf{appr}}(y_i) \subseteq Sat_A^{\mathsf{exact}}(y_i), \quad Sat_E^{\mathsf{exact}}(y_i) \subseteq Sat_E^{\mathsf{appr}}(y_i).$$

Proof: If $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) \in Sat_A^{\mathsf{appr}}(y_i)$, i.e., $\chi_{Sat_A^{\mathsf{appr}}(y_i)}(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) = 1$, then $\lambda_i|_{\vec{q}=\vec{q}^{\mathsf{fix}}, \vec{x}=\vec{x}^{\mathsf{fix}}} = 1$ according to Def. 13, Eqn. (1). Consider an *arbitrary* completion $D^c \in \mathcal{C}(D)$ where the replacement of the Black Boxes introduces the additional state variables $\vec{q}^r \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$ and let $\lambda_i^c((\vec{q}, \vec{q}^r), \vec{x})$ be the i-th output function of D^c. According to Thm. 4, $\lambda_i^c|_{\vec{q}=\vec{q}^{\mathsf{fix}}, \vec{x}=\vec{x}^{\mathsf{fix}}} = 1$, and thus $\lambda_i^c((\vec{q}^{\mathsf{fix}}, \vec{q}^r), \vec{x}^{\mathsf{fix}}) = 1$ for all $\vec{q}^r \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$. That means that $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) \in Sat_A^{\mathsf{exact}}(y_i)$, since D^c was chosen arbitrarily.

If $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) \notin Sat_E^{\mathsf{appr}}(y_i)$, then $\lambda_i|_{\vec{q}=\vec{q}^{\mathsf{fix}}, \vec{x}=\vec{x}^{\mathsf{fix}}} = 0$ according to Def. 13, Eqn. (2). Then for an arbitrary completion D^c as given above we have according to Thm. 4: $\lambda_i^c|_{\vec{q}=\vec{q}^{\mathsf{fix}}, \vec{x}=\vec{x}^{\mathsf{fix}}} = 0$, and thus $\lambda_i^c((\vec{q}^{\mathsf{fix}}, \vec{q}^r), \vec{x}^{\mathsf{fix}}) = 0$ for all $\vec{q}^r \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$. That means that $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}) \notin Sat_E^{\mathsf{exact}}(y_i)$, since D^c was chosen arbitrarily. □

The computation of $Sat_A^{\mathsf{appr}}(\varphi)$ and $Sat_E^{\mathsf{appr}}(\varphi)$ for general CTL formulas φ is performed based on possible transitions. Here we work with an approximation, too. We compute an over-approximation of the possible transitions, represented by the characteristic function $\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}')$:

Definition 14.
$$\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}') := \exists \vec{Z}_l \Big(\prod_{i=1}^{|\vec{q}|} \exists Z \big(\delta_i(\vec{q}, \vec{x}, Z, \vec{Z}_l) \equiv q_i'\big) \Big).$$

The following lemma states that χ_{R_E} over-approximates the possible transitions:

Lemma 8. *If $\chi_{R_E}(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, \vec{q}'^{\mathsf{fix}}) = 0$, then there is no possible transition from $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ to $(\vec{q}'^{\mathsf{fix}}, \vec{x}'^{\mathsf{fix}})$ (for an arbitrary next input $\vec{x}'^{\mathsf{fix}}$).*

Proof: If $\chi_{R_E}(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, \vec{q}'^{\mathsf{fix}}) = 0$, then $\forall \vec{Z}_l \Big(\bigvee_{i=1}^{|\vec{q}|} \forall Z \big(\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l) \not\equiv {q_i'}^{\mathsf{fix}}\big) \Big) = 1$. This means that for an arbitrary fixed output $\vec{Z}_l^{\mathsf{fix}}$ of the Black Boxes modeled by Z_i's there is an $i \in \{0, \ldots, |\vec{q}| - 1\}$ with

$$\forall Z \big(\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l^{\mathsf{fix}}) \not\equiv {q_i'}^{\mathsf{fix}}\big) = 1. \tag{3}$$

$\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l^{\mathsf{fix}}) = Z$, $\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l^{\mathsf{fix}}) = {q_i'}^{\mathsf{fix}}$ or $\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l^{\mathsf{fix}}) = \neg {q_i'}^{\mathsf{fix}}$ according to Lemma 2. In the two former cases, Eq. (3) would not hold, thus we have $\delta_i(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}}, Z, \vec{Z}_l^{\mathsf{fix}}) = \neg {q_i'}^{\mathsf{fix}}$, i.e., the output value of δ_i differs from ${q_i'}^{\mathsf{fix}}$ for each replacement of the Z-modeled Black Boxes (Thm. 4).

Altogether we can conclude that the output value of $\vec{\delta}$ for input $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ differs from $\vec{q}'^{\mathsf{fix}}$ independently from the values at the outputs of Black Boxes, i.e., there cannot be a possible transition from $(\vec{q}^{\mathsf{fix}}, \vec{x}^{\mathsf{fix}})$ to $(\vec{q}'^{\mathsf{fix}}, \vec{x}'^{\mathsf{fix}})$. □

Remark. Extending Definitions 11 and 12 with respect to our approximations, we denote in the following not only the states in $Sat_E^{\mathsf{exact}}(\varphi)$, but also the states in $Sat_E^{\mathsf{appr}}(\varphi)$ by 'states *possibly* satisfying φ'. Similarly, we characterize the states in $Sat_A^{\mathsf{appr}}(\varphi)$ by 'states *definitely* satisfying φ' and all transitions described by χ_{R_E} as 'possible transitions'.

Based on χ_{R_E}, $Sat_A^{\mathsf{appr}}(y_i)$ and $Sat_E^{\mathsf{appr}}(y_i)$, it is possible to define rules how arbitrary CTL formulas can be recursively evaluated. We show here how to compute sets $Sat_A^{\mathsf{appr}}(\cdot)$ and $Sat_E^{\mathsf{appr}}(\cdot)$ for CTL formulas $EX\psi$, $\neg\psi$, $(\psi_1 \vee \psi_2)$, $EG\psi$, and $E\psi_1 U \psi_2$. We start with $EX\psi$:

The basic idea behind the definition of $Sat_E^{\mathsf{appr}}(EX\psi)$ and $Sat_A^{\mathsf{appr}}(EX\psi)$ is the following: If there is a possible transition from a state $(\vec{q},\vec{x})$ to another state possibly satisfying ψ, then $(\vec{q},\vec{x})$ possibly satisfies $EX\psi$. If *all* possible transitions from a state $(\vec{q},\vec{x})$ lead to states definitely satisfying ψ, then $(\vec{q},\vec{x})$ definitely satisfies $EX\psi$. The next definition formalizes and refines this idea:

Definition 15.

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

Lemma 9. *Let $Sat_A^{\mathsf{appr}}(\psi)$ be an under-approximation of $Sat_A^{\mathsf{exact}}(\psi)$ and $Sat_E^{\mathsf{appr}}(\psi)$ be an over-approximation of $Sat_E^{\mathsf{exact}}(\psi)$, let $Sat_A^{\mathsf{appr}}(EX\psi)$ and $Sat_E^{\mathsf{appr}}(EX\psi)$ be defined as in Def. 15. Then*

$$Sat_E^{\mathsf{exact}}(EX\psi) \subseteq Sat_E^{\mathsf{appr}}(EX\psi), \quad Sat_A^{\mathsf{appr}}(EX\psi) \subseteq Sat_A^{\mathsf{exact}}(EX\psi).$$

Proof: To prove the first part of Lemma 9, we assume that $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}}) \in Sat_E^{\mathsf{exact}}(EX\psi)$. This implies that there is a possible transition from $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}})$ to some state $(\vec{q}^{\,\prime\mathsf{fix}},\vec{x}^{\,\prime\mathsf{fix}}) \in Sat_E^{\mathsf{exact}}(\psi)$. Since $Sat_E^{\mathsf{exact}}(\psi) \subseteq Sat_E^{\mathsf{appr}}(\psi)$, there is also a possible transition (given by χ_{R_E}) from $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}})$ to some state $(\vec{q}^{\,\prime\mathsf{fix}},\vec{x}^{\,\prime\mathsf{fix}}) \in Sat_E^{\mathsf{appr}}(\psi)$. By Def. 15, this means that $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}}) \in Sat_E^{\mathsf{appr}}(EX\psi)$.

The proof for the second part is slightly more involved: Assume that $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}}) \notin Sat_A^{\mathsf{exact}}(EX\psi)$. That means that there exists a completion D^c of the incomplete design (with transition function $\vec{\delta}^c$, and additional state variables $\vec{q}^{\,r} \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$ introduced by replacements of Black Boxes) and there exists $\vec{q}^{\,\mathsf{fix}^r} \in \mathbb{B}^{|\vec{Q}^c|-|\vec{Q}|}$, such that $((\vec{q}^{\,\mathsf{fix}},\vec{q}^{\,\mathsf{fix}^r}),\vec{x}^{\,\mathsf{fix}})) \not\models EX\psi$. I.e., with $\vec{\delta}^c((\vec{q}^{\,\mathsf{fix}},\vec{q}^{\,\mathsf{fix}^r}),\vec{x}^{\,\mathsf{fix}}) = (\vec{q}^{\,\prime\mathsf{fix}},\vec{q}^{\,\prime\mathsf{fix}^r})$: $((\vec{q}^{\,\prime\mathsf{fix}},\vec{q}^{\,\prime\mathsf{fix}^r}),\vec{x}^{\,\prime\mathsf{fix}}) \not\models \psi\ \forall \vec{x}^{\,\prime\mathsf{fix}} \in \mathbb{B}^{|\vec{X}|}$. This leads to $\chi_{R_E}(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}},\vec{q}^{\,\prime\mathsf{fix}}) = 1 \ \wedge\ \forall \vec{x}^{\,\prime\mathsf{fix}} \in \mathbb{B}^{|\vec{X}|} : (\vec{q}^{\,\prime\mathsf{fix}},\vec{x}^{\,\prime\mathsf{fix}}) \notin Sat_A^{\mathsf{exact}}(\psi)$. Since $Sat_A^{\mathsf{appr}}(\psi) \subseteq Sat_A^{\mathsf{exact}}(\psi)$:
$\chi_{R_E}(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}},\vec{q}^{\,\prime\mathsf{fix}}) = 1 \wedge \forall \vec{x}^{\,\prime\mathsf{fix}} \in \mathbb{B}^{|\vec{X}|} : (\vec{q}^{\,\prime\mathsf{fix}},\vec{x}^{\,\prime\mathsf{fix}}) \notin Sat_A^{\mathsf{appr}}(\psi)$ and thus $\big(\chi_{R_E}(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}},\vec{q}^{\,\prime\mathsf{fix}}) \cdot \forall \vec{x}\,'\overline{\chi_{Sat_A^{\mathsf{appr}}(\psi)}}(\vec{q}^{\,\prime\mathsf{fix}},\vec{x}\,')\big) = 1$. According to Def. 15 this implies $\chi_{Sat_A^{\mathsf{appr}}(EX\psi)}(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}}) = 0$, i.e., $(\vec{q}^{\,\mathsf{fix}},\vec{x}^{\,\mathsf{fix}}) \notin Sat_A^{\mathsf{appr}}(EX\psi)$. □

Negation and disjunction are handled as follows:

Definition 16.

$$\chi_{Sat_A^{\mathsf{appr}}(\neg\psi)}(\vec{q},\vec{x}) := \overline{\chi_{Sat_E^{\mathsf{appr}}(\psi)}}(\vec{q},\vec{x}) \quad \text{and}$$
$$\chi_{Sat_E^{\mathsf{appr}}(\neg\psi)}(\vec{q},\vec{x}) := \overline{\chi_{Sat_A^{\mathsf{appr}}(\psi)}}(\vec{q},\vec{x}),$$
$$\chi_{Sat_A^{\mathsf{appr}}(\psi_1\vee\psi_2)}(\vec{q},\vec{x}) := (\chi_{Sat_A^{\mathsf{appr}}(\psi_1)} \vee \chi_{Sat_A^{\mathsf{appr}}(\psi_2)})(\vec{q},\vec{x}) \quad \text{and}$$
$$\chi_{Sat_E^{\mathsf{appr}}(\psi_1\vee\psi_2)}(\vec{q},\vec{x}) := (\chi_{Sat_E^{\mathsf{appr}}(\psi_1)} \vee \chi_{Sat_E^{\mathsf{appr}}(\psi_2)})(\vec{q},\vec{x}).$$

Note that negation plays a special role here, since it turns an *over-approximation* of the set of states which *possibly* satisfy ψ into an *under-approximation* of the set of states which *definitely* satisfy $\neg\psi$ (and vice versa).

Lemma 10. *Let $Sat_A^{\mathsf{appr}}(\psi)$ be an under-approximation of $Sat_A^{\mathsf{exact}}(\psi)$ and $Sat_E^{\mathsf{appr}}(\psi)$ be an over-approximation of $Sat_E^{\mathsf{exact}}(\psi)$, let $Sat_A^{\mathsf{appr}}(\neg\psi)$, $Sat_E^{\mathsf{appr}}(\neg\psi)$, $Sat_A^{\mathsf{appr}}(\psi_1\vee\psi_2)$, and $Sat_E^{\mathsf{appr}}(\psi_1\vee\psi_2)$ be defined by Def. 16. Then*

$$Sat_A^{\mathsf{appr}}(\neg\psi) \subseteq Sat_A^{\mathsf{exact}}(\neg\psi), \tag{4}$$
$$Sat_E^{\mathsf{exact}}(\neg\psi) \subseteq Sat_E^{\mathsf{appr}}(\neg\psi), \tag{5}$$
$$Sat_A^{\mathsf{appr}}(\psi_1\vee\psi_2) \subseteq Sat_A^{\mathsf{exact}}(\psi_1\vee\psi_2), \tag{6}$$
$$Sat_E^{\mathsf{exact}}(\psi_1\vee\psi_2) \subseteq Sat_E^{\mathsf{appr}}(\psi_1\vee\psi_2). \tag{7}$$

Proof:

$$\begin{aligned} Sat_A^{\mathsf{appr}}(\neg\psi) &= (\mathbb{B}^{|\vec{Q}|}\times\mathbb{B}^{|\vec{X}|}) \setminus Sat_E^{\mathsf{appr}}(\psi) && \text{(Def.16)}\\ &\subseteq (\mathbb{B}^{|\vec{Q}|}\times\mathbb{B}^{|\vec{X}|}) \setminus Sat_E^{\mathsf{exact}}(\psi) && \text{(Precond. Lemma 10)}\\ &= Sat_A^{\mathsf{exact}}(\neg\psi). && \text{(Def.11)} \end{aligned}$$

This proves Eqn. (4). With an analogous argument Eqn. (5) can be proved. Eqn. (6) and Eqn. (7) follow easily by appropriate set operations. □

Finally, we define $\varphi = EG\psi$ and $\varphi = E\psi_1 U \psi_2$ to be evaluated by their standard fixed point iterations (see Figs. 2, 3) based on the evaluation of EX defined above (two separate fixed point iterations for $Sat_A^{\mathsf{appr}}(\cdot)$ and $Sat_E^{\mathsf{appr}}(\cdot)$).

Lemma 11. *If $Sat_A^{\mathsf{appr}}(\psi) \subseteq Sat_A^{\mathsf{exact}}(\psi)$ and $Sat_E^{\mathsf{exact}}(\psi) \subseteq Sat_E^{\mathsf{appr}}(\psi)$, $Sat_A^{\mathsf{appr}}(EG\psi)$, $Sat_E^{\mathsf{appr}}(EG\psi)$, $Sat_A^{\mathsf{appr}}(E\psi_1U\psi_2)$, and $Sat_E^{\mathsf{appr}}(E\psi_1U\psi_2)$ are obtained by the fixed point iteration of Figs. 2 and 3, then $Sat_A^{\mathsf{appr}}(EG\psi) \subseteq Sat_A^{\mathsf{exact}}(EG\psi)$, $Sat_E^{\mathsf{exact}}(EG\psi) \subseteq Sat_E^{\mathsf{appr}}(EG\psi)$, $Sat_A^{\mathsf{appr}}(E\psi_1U\psi_2) \subseteq Sat_A^{\mathsf{exact}}(E\psi_1U\psi_2)$ and $Sat_E^{\mathsf{exact}}(E\psi_1U\psi_2) \subseteq Sat_E^{\mathsf{appr}}(E\psi_1U\psi_2)$.*

Proof: Since the evaluation of EG and EU is done by iterated application of the EX-operator according to Figs. 2 and 3, the proof follows immediately from the corresponding properties of EX (see Lemma 9) and monotonicity of set union resp. intersection. □

Theorem 12. *If $\chi_{Sat_A^{\mathsf{appr}}(\varphi)}$ and $\chi_{Sat_A^{\mathsf{appr}}(\varphi)}$ are computed recursively according to Definitions 13, 15, 16, then*

$$\forall \vec{x}\Big(\big(\chi_{Sat_A^{\mathsf{appr}}(\varphi)}(\vec{q},\vec{x})\big)|_{\vec{q}=\vec{q}^{\,0}}\Big) = 1 \quad\Longrightarrow\quad \varphi \text{ is valid}$$
$$\exists \vec{x}\Big(\overline{\chi_{Sat_E^{\mathsf{appr}}(\varphi)}(\vec{q},\vec{x})}|_{\vec{q}=\vec{q}^{\,0}}\Big) = 1 \quad\Longrightarrow\quad \varphi \text{ is not realizable}$$

Proof: The proof follows directly from Lemmas 6–11. □

Note that the results in this section have a strong relationship to 3-valued model checking known from the context of software model checking [12], [13]. Details are discussed in Sect. 8.

5.3 Including Z_i-Variables into the State Space

Sometimes the approximations considered above are too coarse to obtain definite answers concerning validity or non-realizability of CTL formulas (see also Sect. 7). A further improvement on the accuracy of the two approximated sets can be obtained by including Z_i-variables assigned to Black Box outputs into the state space.

As a motivation for this, consider the CTL formula $\varphi = (y_1 \rightarrow EXy_3) = (\neg y_1 \vee EXy_3)$ for the design illustrated in Fig. 1 b). The formula essentially says that if the first output Y_1 holds the value 1 in the initial state, then the third output Y_3 holds the value 1 in the next state of the design. The formula is valid, since 'output Y_1 is 1' implies that the flip flop input is 1 and thus output Y_3 is 1 in the next state. This holds independently from the implementation of the Black Box.

If we recursively compute $\chi_{Sat_A^{\mathsf{appr}}(\neg y_1 \vee EXy_3)}$ according to the previous section (modeling the Black Box output by Z_1), we obtain:

$$\begin{aligned} \chi_{Sat_A^{\mathsf{appr}}(\neg y_1)}(q_1,x_1) &= \forall Z_1(\overline{\lambda_1}) = \overline{x}_1\\ \chi_{Sat_A^{\mathsf{appr}}(y_3)}(q_1,x_1) &= \forall Z_1(\lambda_3) = q_1\\ \chi_{Sat_A^{\mathsf{appr}}(EXy_3)}(q_1,x_1) &= \forall q_1'\Big(\big(\exists Z_1(\delta_1 \equiv q_1')\big) \rightarrow \exists x_1'(\chi_{Sat_A^{\mathsf{appr}}(y_3)})|_{\substack{q_1\leftarrow q_1'\\ x_1\leftarrow x_1'}}\Big)\\ &= \forall q_1'\Big(\big(\exists Z_1(Z_1 \equiv q_1')\big) \rightarrow q_1'\Big) = 0. \end{aligned}$$

The validity of φ *cannot* be shown, since

$$\forall x_1(\chi_{Sat_A^{\mathsf{appr}}(y_1\vee EXy_3)}|_{q_1=q_1^0}) = \forall x_1(\overline{x}_1 \vee 0) = 0.$$

Having a closer look at the computation above, we observe: On the one hand, only those states with $x_1 = 0$ are included into $Sat_A^{\mathsf{appr}}(\neg y_1)$, since the output of the Black Box output might be 1 in the current state. On the other hand, $Sat_A^{\mathsf{appr}}(EXy_3)(q_1,x_1) = \varnothing$, since the output of the Black Box output might be 0 in the current state. Clearly the Black Box output cannot be 0 and 1 at the same time and by case distinction wrt. Z_1 we can prove that φ is valid.

Based on this consideration, our model checking routine for incomplete designs is improved by including Z_i-variables assigned to Black Box outputs into the state space. In this way the corresponding Black Box output values Z_i are constant within each single state and therefore in our example Z_1 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 $\vec{\lambda}(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)$ and the transition function $\vec{\delta}(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)$ can be computed by using the symbolic simulation from Sect. 4 for which it is not necessary to distinguish between $\vec{Z}_l$ and $\vec{Z}_o$.

Now the computation of sets $Sat_A^{\mathsf{appr,incl}}(\cdot)$ and $Sat_E^{\mathsf{appr,incl}}(\cdot)$ is performed in a manner similar to the previous section. We start with the sets of states definitely or possibly satisfying the atomic CTL formula y_i:

Definition 17.

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

Lemma 13. *For $Sat_A^{\mathsf{appr,incl}}(y_i)$, $Sat_E^{\mathsf{appr,incl}}(y_i)$ as defined in Def. 17:*

$$\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(y_i)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(y_i)}, \quad \chi_{Sat_E^{\mathsf{exact}}(y_i)} \le \big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(y_i)}\big).$$

Proof: The proof follows immediately from Lemma 7, since $\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(y_i)}\big) = \chi_{Sat_A^{\mathsf{appr}}(y_i)}$ and $\big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(y_i)}\big) = \chi_{Sat_E^{\mathsf{appr}}(y_i)}$. □

Analogously, we define an over-approximation $\chi_{R_E^{\mathsf{incl}}}$ for the characteristic function of possible transitions:

Definition 18.

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

As in the previous section the sets $Sat_A^{\mathsf{appr,incl}}(EX\psi)$ and $Sat_E^{\mathsf{appr,incl}}(EX\psi)$ are computed based on $Sat_A^{\mathsf{appr,incl}}(\psi)$, $Sat_E^{\mathsf{appr,incl}}(\psi)$ and $\chi_{R_E^{\mathsf{incl}}}$:

Definition 19.

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

Lemma 14. *If both $Sat_A^{\mathsf{appr,incl}}(EX\psi)$ and $Sat_E^{\mathsf{appr,incl}}(EX\psi)$ are defined as in Def. 19, $\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(\psi)}$ and $\big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(\psi)}\big) \ge \chi_{Sat_E^{\mathsf{exact}}(\psi)}$, then*

$$\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(EX\psi)} \text{ and}$$
$$\chi_{Sat_E^{\mathsf{exact}}(EX\psi)} \le \big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(EX\psi)}\big).$$

Proof:

Part 1: $\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)}\big)$

$$= \forall \vec{Z}_o \forall \vec{q}\,'\Big(\big(\neg \chi_{R_E^{\mathsf{incl}}}\big) \vee \exists \vec{x}\,' \forall \vec{Z}_o' \big(\chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z}_o \leftarrow \vec{Z}_o'}}\big)\Big)$$
$$= \forall \vec{q}\,'\Big(\big(\neg \underbrace{\exists \vec{Z}_o \chi_{R_E^{\mathsf{incl}}}}\big) \vee \exists \vec{x}\,' \underbrace{\forall \vec{Z}_o' \big(\chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z}_o \leftarrow \vec{Z}_o'}}\big)\Big)$$

From Definitions 14 and 18 we conclude $\big(\exists \vec{Z}_o \chi_{R_E^{\mathsf{incl}}}\big) = \chi_{R_E}$ and from the precondition of the lemma we know

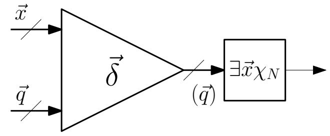


Fig. 6. Illustration for the functional preimage computation for complete designs.

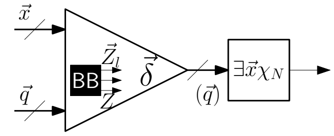


Fig. 7. Illustration for the functional preimage computation for incomplete designs

$\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(\psi)}$. So we can apply Lemma 9 and obtain $\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(EX\psi)}$.

Part 2: $\big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(EX\psi)}\big) =$

$$= \exists \vec{Z}_o \exists \vec{q}\,'\Big(\chi_{R_E^{\mathsf{incl}}} \cdot \exists \vec{x}\,' \exists \vec{Z}_o' \big(\chi_{Sat_E^{\mathsf{appr,incl}}(\psi)}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z}_o \leftarrow \vec{Z}_o'}}\big)\Big)$$
$$= \exists \vec{q}\,'\Big(\underbrace{\exists \vec{Z}_o \chi_{R_E^{\mathsf{incl}}}} \cdot \exists \vec{x}\,' \underbrace{\exists \vec{Z}_o' \big(\chi_{Sat_E^{\mathsf{appr,incl}}(\psi)}}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z}_o \leftarrow \vec{Z}_o'}}\big)\Big)$$

Again, we apply Defs. 14, 18, the precondition of the lemma and Lemma 9 and obtain $\big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(EX\psi)}\big) \ge \chi_{Sat_E^{\mathsf{exact}}(EX\psi)}$. □

For all remaining CTL operators $\neg$, $\vee$, EG and EU, $Sat_A^{\mathsf{appr,incl}}(\cdot)$ and $Sat_E^{\mathsf{appr,incl}}(\cdot)$ are computed as already described in Sect. 5.2. Lemmas like Lemma 10 and 11 hold with exactly the same arguments as in Sect. 5.2.

Theorem 15. *If $\chi_{Sat_A^{\mathsf{appr,incl}}(\varphi)}$ and $\chi_{Sat_E^{\mathsf{appr,incl}}(\varphi)}$ are computed recursively as described above, then*

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

Proof: As shown above we have $\big(\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\varphi)}\big) \le \chi_{Sat_A^{\mathsf{exact}}(\varphi)}$ and $\big(\exists \vec{Z}_o \chi_{Sat_E^{\mathsf{appr,incl}}(\varphi)}\big) \ge \chi_{Sat_E^{\mathsf{exact}}(\varphi)}$. So the theorem follows from Lemma 6. □

Example. Again, we consider the CTL formula $\varphi = (\neg y_1 \vee EXy_3)$ for the design illustrated in Fig. 1 b):

$$\chi_{Sat_A^{\mathsf{appr,incl}}(\neg y_1)}(q_1, x_1, Z_1) = \overline{\lambda_1} = \overline{x}_1 \vee \overline{Z}_1$$
$$\chi_{Sat_A^{\mathsf{appr,incl}}(y_3)}(q_1, x_1, Z_1) = \lambda_3 = q_1$$
$$\chi_{Sat_A^{\mathsf{appr,incl}}(EXy_3)}(q_1, x_1, Z_1) = \forall q_1'\big((\delta_1 \equiv q_1') \to \exists x_1' \forall Z_1' (\chi_{Sat_A(y_3)})|_{\substack{q_1 \leftarrow q_1' \\ x_1 \leftarrow x_1' \\ Z_1 \leftarrow Z_1'}}\big)$$
$$= \forall q_1'\big((Z_1 \equiv q_1') \to q_1'\big) = Z_1$$

Now the validity of φ can be shown:

$$\forall x_1 \forall Z_1 (\chi_{Sat_A^{\mathsf{appr,incl}}(y_1 \vee EXy_3)}|_{q_1 = q_1^0}) = \forall x_1 \forall Z_1 (\overline{x}_1 \vee \overline{Z}_1 \vee Z_1) = 1$$

5.4 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 $Sat(EX\psi)$ [22], [23]:

So far, we used the *relational* approach in our approximate model checking procedures for incomplete designs. For complete designs this approach builds the characteristic function of the transition relation $\chi_R(\vec{q}, \vec{x}, \vec{q}\,') := \prod_{i=1}^{|\vec{q}|} \big(\delta_i(\vec{q}, \vec{x}) \equiv q_i'\big)$ which is then used in the actual preimage computation for a given set of states (represented by χ_N in this case):

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

The *functional* approach uses the *compose* operator, with $f|_{x_i \leftarrow g} := \overline{g} \cdot f|_{x_i=0} + g \cdot f|_{x_i=1}$ for $f, g\colon \mathbb{B}^n \to \mathbb{B}$ and an input variable x_i of f (see Def. 5). Based on the compose operator, the preimage of a set of states given by χ_N can be computed as follows:

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

(Here the composition of different variables q_i by functions $\vec{\delta}(\vec{q}, \vec{x})$ is performed in parallel.) Fig. 6 illustrates this composition as the composition of the Boolean circuit for $\vec{\delta}$ into the Boolean circuit for the characteristic function $\big(\exists\vec{x}\chi_N\big)(\vec{q})$ (variables q_i of $\big(\exists\vec{x}\chi_N\big)(\vec{q})$ are replaced by the corresponding transition functions $\delta_i(\vec{q},\vec{x})$).

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 this, the functional version of preimage computation is often more efficient than the relational version [22].

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 that Z_i-variables are *not* included in the state space in order to keep the presentation compact.

Taking into account that the transition function $\vec{\delta}(\vec{q},\vec{x},Z,\vec{Z}_l)$ now depends on the additional variables Z and $\vec{Z}_l$, we have to replace the usual *compose* operator by a new *compose-Z* operator:

Definition 20. *The* compose-Z *operator "$|^{cZ}$" for $f\colon \mathbb{B}^n \to \mathbb{B}$ with input variables $y_1,\dots,y_n$ and $g\colon \mathbb{B}^{n+1} \to \mathbb{B}$ with input variables $y_1,\dots,y_n$, Z, is defined as:*

$$f|^{cZ}_{y_i\leftarrow g} := \overline{g}|_{Z\leftarrow\overline{Z}} \cdot f|_{y_i=0} + g \cdot f|_{y_i=1} \tag{8}$$

Just as in the definition of symbolic Z-simulation in Sect. 4 we have to replace Z by $\overline{Z}$ after negation in the formula for *compose-Z*.

A composition of a *vector* of variables by *compose-Z* is computed by a recursive computation of compositions (as for the original *compose* operator) and the formula $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)|_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x})}$ for the complete case is now replaced by $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)|^{cZ}_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x},Z,\vec{Z}_l)}$ for the incomplete case.

For a better understanding of *compose-Z* (together with its deficiencies explained in the following and a corresponding improvement) please assume for a moment that $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)$ is represented as a BDD which in turn can be seen as a multiplexer circuit. In Fig. 7 the output functions $\delta_i(\vec{q},\vec{x},Z,\vec{Z}_l)$ of $\vec{\delta}$ are inputs to the BDD for $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)$, i.e., they correspond to select-inputs of the multiplexers in the circuit representation. Now it is easy to see that $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)|^{cZ}_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x},Z,\vec{Z}_l)}$ as defined above can be interpreted as the result of a symbolic Z/Z_i-simulation of the Boolean circuit given in Fig. 7 (δ_i play the role of g in Eqn. (8), the outputs of multiplexers in $\big(\exists\vec{x}\chi_N(\vec{q},\vec{x})\big)$ play the role of f in Eqn. (8)).

However, there is an obvious deficiency of simple symbolic Z/Z_i-simulation applied to multiplexer circuits derived from BDDs: Consider the case that $\big(f|_{y_i=0}\big)|_{\vec{y}=\vec{\epsilon}} = 1$, $\big(f|_{y_i=1}\big)|_{\vec{y}=\vec{\epsilon}} = 1$ and $g|_{\vec{y}=\vec{\epsilon}} = Z$ in Eqn. (8). Symbolic Z/Z_i-simulation computes $\big(f|^{cZ}_{y_i\leftarrow g}\big)|_{\vec{y}=\vec{\epsilon}} = Z$. In this special case it is easy to see that this is an inaccuracy (inherited from conventional $(0,1,X)$-simulation) which is not really needed: $g|_{\vec{y}=\vec{\epsilon}}$ selects between $\big(f|_{y_i=0}\big)|_{\vec{y}=\vec{\epsilon}} = 1$ and $\big(f|_{y_i=1}\big)|_{\vec{y}=\vec{\epsilon}} = 1$. Even if the value of $g|_{\vec{y}=\vec{\epsilon}}$ is unknown, we can easily conclude that the output of $\big(f|^{cZ}_{y_i\leftarrow g}\big)|_{\vec{y}=\vec{\epsilon}}$ is 1. Based on this observation we define an improved *compose-Z* operator which improves the accuracy of the simple symbolic Z/Z_i-simulation by replacing Eqn. (8) as follows:

Definition 21. *The improved compose-Z operator "$|^{cZ,\mathsf{impr}}$" for $f\colon \mathbb{B}^n \to \mathbb{B}$ with input variables $y_1,\dots,y_n$ and $g\colon \mathbb{B}^{n+1} \to \mathbb{B}$ with input variables $y_1,\dots,y_n$, Z, is defined as:*

$$f|^{cZ,\mathsf{impr}}_{y_i\leftarrow g} := \overline{g}|_{Z\leftarrow\overline{Z}} \cdot f|_{y_i=0} + g \cdot f|_{y_i=1} + f|_{y_i=0} \cdot f|_{y_i=1} \tag{9}$$

The additional term in (9) may seem to be unnecessary at first sight, yet it is easy to see that for $\big(f|_{y_i=0}\big)|_{\vec{y}=\vec{\epsilon}} = 1$, $\big(f|_{y_i=1}\big)|_{\vec{y}=\vec{\epsilon}} = 1$ and $g|_{\vec{y}=\vec{\epsilon}} = Z$ Eqn. (9) results in $\big(f|^{cZ,\mathsf{impr}}_{x_i\leftarrow g}\big)|_{\vec{y}=\vec{\epsilon}} = 1$ which is more exact than the result of simple Z/Z_i-simulation (i.e., the result contains more accurate information).

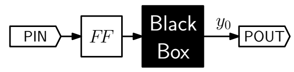


Fig. 8. Motivating Example for Exact Symbolic Model Checking

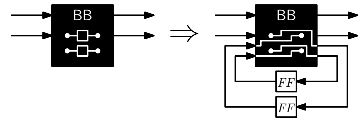


Fig. 9. Extracting the flip-flops from a Black Box with bounded memory size of 2.

For the case that more than one variable is replaced by a function, we have to define composition recursively:

Definition 22. *Let $f\colon \mathbb{B}^n \to \mathbb{B}$ be a Boolean function over variables $\vec{y} = (y_1,\dots,y_n)$ and $g_i\colon \mathbb{B}^{n+1} \to \mathbb{B}$ $(1 \le i \le k)$ be Boolean functions over variables $(y_1,\dots,y_n,Z)$. The improved compose-Z operator "$|^{cZ,\mathsf{impr}}$" is then recursively defined as:*

$$\begin{aligned}
f|^{cZ,\mathsf{impr}}_{\substack{y_1\leftarrow g_1\\ \dots\\ y_k\leftarrow g_k}} :=\ & \overline{g}_k|_{Z\leftarrow\overline{Z}} \cdot \big(f|_{y_k=0}\big)|^{cZ,\mathsf{impr}}_{\substack{y_1\leftarrow g_1\\ \dots\\ y_{k-1}\leftarrow g_{k-1}}} + g_k \cdot \big(f|_{y_k=1}\big)|^{cZ,\mathsf{impr}}_{\substack{y_1\leftarrow g_1\\ \dots\\ y_{k-1}\leftarrow g_{k-1}}}\\
& + \big(f|_{y_k=0}\big)|^{cZ,\mathsf{impr}}_{\substack{y_1\leftarrow g_1\\ \dots\\ y_{k-1}\leftarrow g_{k-1}}} \cdot \big(f|_{y_k=1}\big)|^{cZ,\mathsf{impr}}_{\substack{y_1\leftarrow g_1\\ \dots\\ y_{k-1}\leftarrow g_{k-1}}}\\
f|^{cZ,\mathsf{impr}}_{\varnothing} :=\ & f
\end{aligned}$$

(Note that composition is performed in parallel here; a straightforward reduction of the composition for k variables to a series of k compositions for single variables would lead to a different result, since functions g_j may depend on replaced variables y_i.)
The consideration given above can be extended to the case that some Black Box outputs are modeled by $\vec{Z}_o$ variables in the state space. Then, using the improved *compose-Z* operator, we can define $\chi_{Sat_A^{\mathsf{func}}(EX\psi)}$ and $\chi_{Sat_E^{\mathsf{func}}(EX\psi)}$ by functional preimage computation:

Definition 23.

$$\chi_{Sat_A^{\mathsf{func}}(EX\psi)}(\vec{q},\vec{x},\vec{Z}_o) := \forall\vec{Z}_l\forall Z\Big(\big(\exists\vec{x}\forall\vec{Z}_o\chi_{Sat_A^{\mathsf{func}}(\psi)}\big)|^{cZ,\mathsf{impr}}_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)}\Big)$$
$$\chi_{Sat_E^{\mathsf{func}}(EX\psi)}(\vec{q},\vec{x},\vec{Z}_o) := \exists\vec{Z}_l\exists Z\Big(\big(\exists\vec{x}\exists\vec{Z}_o\chi_{Sat_E^{\mathsf{func}}(\psi)}\big)|^{cZ,\mathsf{impr}}_{\vec{q}\leftarrow\vec{\delta}(\vec{q},\vec{x},Z,\vec{Z}_l,\vec{Z}_o)}\Big)$$

For all other operators $\chi_{Sat_E^{\mathsf{func}}(\varphi)}$ and $\chi_{Sat_A^{\mathsf{func}}(\varphi)}$ are defined just as $\chi_{Sat_E^{\mathsf{appr,incl}}(\varphi)}$ and $\chi_{Sat_A^{\mathsf{appr,incl}}(\varphi)}$.

Interestingly, we are able to prove that functional preimage computation with the improved *compose-Z* operator as defined above gives *exactly the same* results as relational preimage computation:

Theorem 16. *For arbitrary CTL formulas φ:*

$$\chi_{Sat_E^{\mathsf{func}}(\varphi)} = \chi_{Sat_E^{\mathsf{appr,incl}}(\varphi)} \text{ and } \chi_{Sat_A^{\mathsf{func}}(\varphi)} = \chi_{Sat_A^{\mathsf{appr,incl}}(\varphi)}.$$

Proof: The proof is performed by induction on the structure of φ. The main part consists of the equivalence of the relational and functional definitions for the EX-operator. This is proven in Appendix D by induction on the number of state bits in the design. □

Experiments showing advantages of model checking using the improved compose-Z operator instead of the relational approach are given in Sect. 7.

6 Exact Symbolic Model Checking for Black Boxes with Bounded Memory

6.1 Motivation

In the last section, we introduced a method to approximate both $Sat_E^{\mathsf{exact}}(\varphi)$, the set of states, for which there is at least one Black Box replacement so that φ is satisfied, and $Sat_A^{\mathsf{exact}}(\varphi)$, the set of states, for which φ is satisfied for all Black Box replacements. Experiments in Sect. 7 show that, based on these sets, we are 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.

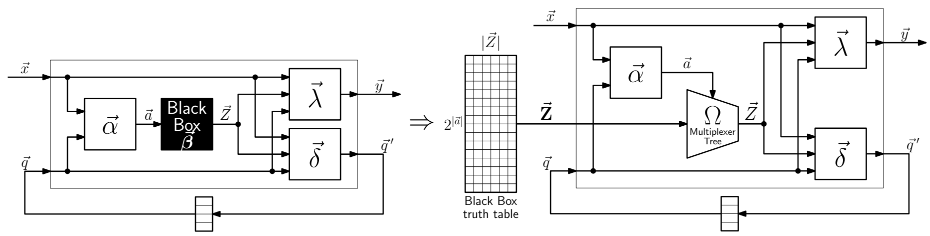


Fig. 10. Incomplete design with one combinational Black Box and the modified design in which the Black Box has been replaced by its truth table variables and a select function.

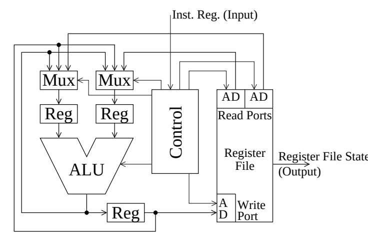


Fig. 11. Pipelined ALU

For completeness we present in this section a concept how to compute an *exact* solution to a restricted problem by means of a *conventional symbolic model checker*: Here we assume that there is a fixed upper bound on the number of flip-flops the possible substitutions of the Black Boxes are allowed to have. Only with this ‘bounded memory assumption’, the number of different possible behaviors of the Black Boxes is finite and thus, it is conceptually possible to compute the set of states fulfilling φ for each possible completion $c \in \mathcal{C}(D)$ of the incomplete design D. (Without the bounded memory assumption the problem with multiple Black Boxes is undecidable [9].)

It is important to note that our method takes into account that the Black Boxes are only able to read the input signals connected to them. Thus, the exact method we present in the following is able to provide an exact answer for the case that the Black Boxes do not have *global knowledge*.

Example. Considering the small example from Fig. 8 together with formula $\varphi = EF(EXy_0 \wedge EX\neg y_0)$, it is easy to see that our approximate methods are neither able to prove that φ is valid nor able to prove that φ is not realizable. However, φ is indeed not realizable (no matter how much memory is used for the Black Box): 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, 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) is the same for both input sequences. This means that the CTL formula φ is not satisfied for any possible Black Box substitutions, thus it is not realizable. Note that synthesis approaches such as [8] would consider it as realizable due to their implicit assumption that the Black Box behavior may depend on all signals of the design.

For most examples an explicit approach enumerating all possible Black Box substitutions is obviously not applicable in practice due to the enormous number of possible substitutions; a confirmation for this statement is given by our experimental results in Sect. 7. For that reason we use *symbolic methods* to implicitly consider all possible choices for the Black Box substitutions in parallel.

First we show how Black Boxes with bounded memory can be transformed into combinational Black Boxes, i.e., Black Boxes that may only be substituted by designs without flip-flops. Then we take a look at a concept for *exact symbolic model checking* for designs containing combinational Black Boxes.

6.2 Exact Algorithm

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 $b \in \mathbb{N}_0$ be this upper bound.

Extracting Flip-Flops

Given this assumption, we can separate the flip-flops from the Black Box without changing the behavior: We have to add b flip-flops to the design and connect them to b additional outputs and b additional inputs of the Black Box as illustrated in Fig. 9 for a Black Box with bounded memory size of 2. The resulting transformed Black Box is combinational, i.e., the possible substitutions are limited to combinational designs.

Now it is sufficient to solve the model checking problem for combinational Black Boxes.

A Concept for Exact Symbolic Model Checking of Incomplete Designs with Combinational Black Boxes

For the time being, we restrict our view to incomplete designs containing exactly *one* Black Box. Having performed the transformation given above we can assume that the Black Box is combinational. Then we can divide the combinational part of the design into four parts (see left part of Fig. 10):

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 $\vec{\beta}\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$. Furthermore, let $\vec{\alpha}\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \to \mathbb{B}^{|\vec{a}|}$ be the Boolean function of the circuit part computing the Black Box inputs $\vec{a}$. As usual, $\vec{\lambda}$ and $\vec{\delta}$ compute the primary outputs resp. the next states. While $\vec{\alpha}$ just depends on the primary input $\vec{x}$ and the current state $\vec{q}$, $\vec{\delta}$ and $\vec{\lambda}$ 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 reduce the question whether there exists a Boolean function $\vec{\beta}$ so that φ is satisfied (realizability) and the question whether φ is satisfied for all Boolean functions $\vec{\beta}$ (validity) to existential resp. universal quantification 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 $\vec{\beta}\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$ 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 $\mathbf{Z}_{i,j}$ ($0 \le i < 2^{|\vec{a}|}$, $0 \le j < |\vec{Z}|$). 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 Boolean 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 right part of Fig. 10). Formally, $\Omega_i(\vec{a}, \vec{\mathbf{Z}}) := \mathbf{Z}_{a,i}$, where 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.)

Definition 24. *The output function* $\vec{\boldsymbol{\lambda}}$ *and the transition function* $\vec{\boldsymbol{\delta}}$ *are defined by*

$$\vec{\boldsymbol{\lambda}}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) := \vec{\lambda}\big(\vec{q}, \vec{x}, \Omega\big(\vec{\alpha}(\vec{q}, \vec{x}), \vec{\mathbf{Z}}\big)\big)$$

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

For our exact symbolic model checking, we essentially perform conventional symbolic model checking (see Sect. 2) based on $\vec{\boldsymbol{\lambda}}$ and $\vec{\boldsymbol{\delta}}$ 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 $\vec{\mathbf{Z}}$.

Theorem 17. *The set of states $\chi_{Sat(\varphi)}(\vec{q},\vec{x},\vec{\mathbf{Z}})$ that satisfy the property φ depending on the Black Box truth table $\vec{\mathbf{Z}}$ can be evaluated as follows:*

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

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

Proof: Since $\vec{\mathbf{Z}}$ represents the complete truth table of the Black Box $\vec{\beta}$, thus its whole functionality, there is a substitution of $\vec{\beta}$ 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. 10). Likewise, a property is satisfied in $(\vec{q}^0,\vec{x})$ for all substitutions of $\vec{\beta}$ 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 $\vec{\mathbf{Z}}$. □

Multiple Black Boxes

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

Complexity

The number of new Boolean variables introduced by the transformation shown above is exponential in the number of Black Box inputs and in the upper bound on the number of flip-flops allowed for Black Box substitutions (according to the bounded memory assumption). For the transformed design we perform conventional symbolic model checking. If in the worst case there is no benefit from the usage of symbolic BDD representations, symbolic model checking needs exponential time measured in the input size, thus the overall run time is double exponential in the worst case. (However, experiments in Sect.7.2 show that we do profit from symbolic representations in practice.)

7 Experimental Results

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 [24] and uses 'Lazy Group Sifting' [25], a reordering technique particularly suited for model checking. Sifting is invoked automatically as soon as the number of active BDD nodes exceeds a (dynamic) threshold. Additionally, partitioned transition functions [26] were used for relational preimage computation. All experiments were performed on a Dual Opteron 2GHz with 4GB RAM under Linux.

7.1 Approximate Model Checking for Incomplete Designs

As a case study we used a class of simple synchronous pipelined ALUs (see Fig. 11) with a register file and a forwarding unit; the circuit is based on the design used in [5]. The ALU itself is 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

$$\varphi = AG\Big(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow \bigwedge_j \big((AX)^2\mathbf{R}_{0,j} \oplus (AX)^2\mathbf{R}_{1,j} \equiv (AX)^3\mathbf{R}_{2,j}\big)\Big)$$

which corresponds to formula (1) in [5].[4] 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 are identical to the exclusive-or of $\mathbf{R}_0$ and $\mathbf{R}_1$ in the state two clock cycles in the future ($\mathbf{R}_{i,j}$ is the value of the j-th Bit of the i-th register in the register file).

Experiment 1 (Proofs of Non-Realizability): In a first experiment, we inserted an error to the implementation of the XOR operation[5], 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 of the adder function, the multiplier function and the registers replaced by Black Boxes.

On the left hand side of Tab. 1 we give results of checking property φ both for complete and incomplete pipelined ALUs with varying word widths. 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 12000 CPU seconds. For this experiment, transition relation based preimage computation was used.

Mainly because 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. 1, columns 2–5). In order to prevent the assumption that this behavior is due to a poor implementation of our basic symbolic model checker, we also include run times produced by the BDD based symbolic CTL model checker implemented in VIS [10] (using 'Lazy Group Sifting' as in our approach). Here only the complete pipelined ALU with a word width of 2 could be verified.

For the incomplete pipelined ALUs we observed the result that even our weakest method for approximate model checking (using symbolic Z-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. 1, 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. 1, 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 to 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, i.e., in the incomplete design they are connected to state variables occurring in the property and thus *cannot* be removed by Cone-of-Influence reduction [27].

Experiment 2 (Proofs of Validity): In a second experiment we

4. $(AX)^2$ is an abbreviation of $AXAX$ and $(AX)^3$ is an abbreviation of $AXAXAX$

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

	Faulty pipelined ALU, Black Box outputs modeled by Z													Correct pipelined ALU, Black Box outputs modeled by Z_i's in the state space												
	No Black Boxes					Adder and multiplier replaced (Z)				Adder, multiplier and 12 registers replaced (Z)				No Black Boxes					Adder and multiplier replaced (Z_i)				Adder, multiplier and 12 registers replaced (Z_i)			
word width	BDD vars	mem. used	BDD nodes	time	VIS time	BDD vars	mem. used	BDD nodes	time	BDD vars	mem. used	BDD nodes	time	BDD vars	mem. used	BDD nodes	time	VIS time	BDD vars	mem. used	BDD nodes	time	BDD vars	mem. used	BDD nodes	time
2	117	30M	201K	8.9	5747	117	16M	87K	4.5	69	12M	26K	0.9	117	18M	186K	8.3	8416	121	13M	50K	2.4	97	13M	52K	1.8
4	193	42M	407K	69.9	TO	193	19M	101K	8.6	97	11M	15K	1.0	193	43M	428K	57.3	TO	201	28M	75K	4.0	153	14M	57K	5.0
6	269	74M	1349K	356.7	TO	269	44M	115K	16.0	125	14M	22K	1.5	269	81M	1386K	395.4	TO	281	30M	61K	8.2	209	28M	70K	4.5
8	345	239M	6295K	2781	TO	345	47M	91K	13.9	153	16M	21K	1.9	TO				TO	361	40M	88K	12.9	265	27M	90K	14.5
12	TO				TO	497	44M	83K	24.9	209	27M	34K	4.8	TO				TO	521	48M	116K	33.9	377	34M	81K	20.5
16	TO				TO	649	48M	88K	47.1	265	36M	28K	5.4	TO				TO	681	48M	135K	59.1	489	48M	98K	35.7
24	TO				TO	953	45M	94K	91.3	377	40M	36K	12.1	TO				TO	1001	45M	90K	83.8	713	46M	113K	66.4
32	TO				TO	1257	54M	216K	232.2	489	47M	35K	17.5	TO				TO	1321	44M	150K	207.7	937	45M	91K	83.0
48	TO				TO	1865	62M	143K	493.1	713	48M	46K	45.1	TO				TO	1961	66M	165K	457.3	1385	50M	152K	175.2
64	TO				TO	2473	65M	167K	3031	937	44M	47K	82.3	TO				TO	2601	67M	183K	2284	1833	62M	160K	287.7

TABLE 1
Pipelined ALU with 16 registers: Falsifying realizability / proving validity of
$AG(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow ((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2))$ using transition relations.

	Outputs of the Black Boxes in Register File modeled with...															
	...separate Z_i variables *in* the state space, using transition relations				...separate Z_i variables *not* in the state space, using transition relations				...one single Z variable *not* in the state space, using transition relations				...one single Z variable *not* in the state space, using compose-Z			
word width	BDD vars	memory used	BDD nodes	time	BDD vars	memory used	BDD nodes	time	BDD vars	memory used	BDD nodes	time	BDD vars	memory used	BDD nodes	time
2	605	578M	13M	28945	605	24M	50K	16.8	101	19M	98K	7.6	67	16M	85K	3.9
4	1141	628M	14M	71524	1141	35M	92K	63.5	133	32M	87K	7.9	85	16M	69K	4.2
6	TO				1677	45M	116K	115.0	165	36M	169K	17.4	103	17M	67K	2.9
8	TO				2213	55M	136K	192.0	197	34M	89K	8.1	121	34M	102K	7.9
12	TO				3285	74M	139K	184.8	261	46M	152K	26.2	157	31M	91K	6.0
16	TO				4357	93M	155K	257.6	325	48M	129K	26.5	193	42M	96K	8.5
24	TO				6501	216M	174K	331.9	453	48M	147K	45.6	265	40M	106K	16.8
32	TO				8645	220M	260K	603.1	581	51M	103K	60.6	337	48M	86K	15.2
48	TO				12933	249M	356K	725.8	837	45M	126K	100.9	481	49M	126K	58.7
64	TO				17221	564M	449K	1278	1093	55M	113K	141.1	625	47M	147K	77.9

TABLE 2
(Correct) incomplete pipelined ALU with 256 registers: Proving the validity of
$AG(\text{“}\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1\text{”} \rightarrow ((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2))$ 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.

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.

On the right hand side of Tab. 1 we give the results for both complete and incomplete pipelined ALUs tested with φ. Again, the timeout was 12000 seconds and preimages were computed using a transition relation.

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 on the left hand side and the right hand side of Tab. 1); 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.

Experiment 3 (Proofs of Validity, Effect of Mixed Z/Z_i, Z_i Inside / Outside State Space): 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.

In Experiment 3.1. 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: In Experiment 3.2. we reduced the accuracy for the Black Boxes in the register file by removing the corresponding Z_i's from the state space, while keeping the ones for the Black Boxes replacing the adder and the multiplier. In Experiment 3.3 a further reduction of accuracy for the Black Boxes in the register file was achieved by modeling their outputs with the single variable Z. Finally, Experiment 3.4 evaluates functional preimage computation in the setup of Experiment 3.3..

In Tab. 2, columns 2–13, we give the results for the incomplete pipelined ALUs with varying word widths tested with φ. Except for the timeouts (the timeout was 86400 seconds (= 1 day)), we always were able to prove validity of φ for the incomplete designs.[6] In the following we discuss the results for this set of experiments:

Experiment 3.1: If all Black Boxes are modeled with Z_i's in the state space, a complex transition relation has to be built 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. 2, columns 2–5).

Experiment 3.2: 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 smaller state space representations, which leads to the

6. Note that 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) leads to the situation that we are *not* able to prove validity of φ. Due to lack of space we do not give run times for these unsuccessful configurations in Tab. 2.

result that we are able to prove validity for all instances within the time limit (see columns 6–9 of Tab. 2).

Experiment 3.3: In the case that all Black Box outputs in the register file are modeled using one single Z variable (columns 10–13 of Tab. 2), there is a further significant decrease in the number of necessary BDD variables. For this reason, there is a considerable 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.

Experiment 3.4: Here the relational preimage computation in the setup of Experiment 3.3 was replaced by functional preimage computation based on the improved *compose-*Z operator as introduced in Sect. 5.4. The results are given in columns 14–17 of Tab. 2. Compared to the corresponding results for preimage computation based on transition relations (columns 10–13) they clearly show that the functional approach using *compose-*Z performs even better in this case.

Additional Experiments: Additional experiments broadening the experimental evaluation can be found in Appendix F.

For the pipelined ALU we also checked two additional formulas from [5] (which are neither universal nor existential). The results confirm our observations already made in this section. We additionally observe that VIS with non-deterministic signals computes wrong results for these properties (which is not very surprising, since they do not belong to ACTL), whereas our tool gives a clear classification into "valid", "non-realizable" or "unknown". It it worth to note that our tool computes these answers within much shorter time and with less memory consumption.

An example modeling a railway system was used as an additional case study. For almost all formulas checked we were able to provide definite results within short computation times. For the set of formulas which fall into the class of safety properties we could show that our approach compares favorably to SAT–based approaches such as [17].

7.2 Exact Symbolic Model Checking for Black Boxes with Bounded Memory

Experiment 4: For a first evaluation of our exact symbolic model checking method that has been presented in Sect. 6, we considered a class of arbiters as described in [6]. Given a resource and a number of clients trying to access the resource, the purpose 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 $\mathrm{req}_1 \ldots \mathrm{req}_n$, with $\mathrm{req}_i = 1$ iff client i requests access to the resource, and n acknowledge outputs $\mathrm{ack}_1 \ldots \mathrm{ack}_n$, with $\mathrm{ack}_i = 1$ iff the arbiter acknowledges the request of client i.

[6] gives three CTL properties that an arbiter for n clients must satisfy in order to work as expected:

$$\varphi_1^n = \bigwedge\nolimits_{1\le i<j\le n} \big(AG\neg(\mathrm{ack}_i \wedge \mathrm{ack}_j)\big)$$
$$\varphi_2^n = \bigwedge\nolimits_{1\le i\le n} \big(AGAF(\mathrm{req}_i \rightarrow \mathrm{ack}_i)\big)$$
$$\varphi_3^n = \bigwedge\nolimits_{1\le i\le n} \big(AG(\mathrm{ack}_i \rightarrow \mathrm{req}_i)\big)$$

Property φ_1^n essentially says that the arbiter does not give an acknowledge to two clients at the same time, φ_2^n states that every persistent request should be eventually acknowledged and φ_3^n checks that no acknowledge is given without a corresponding request.

For an arbiter with n clients, [6] provides an implementation which 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 $\mathrm{req}_1 \ldots \mathrm{req}_n$, n outputs $\mathrm{ack}_1 \ldots \mathrm{ack}_n$ and a bounded memory of size $m < 2n$. 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.

Note that the approximate methods as given in Sect. 5 would not be able to provide any proof, since the property φ^n is realizable, but not valid.

Considering an arbiter for 2 clients, the implementation given in [6] 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 [6] has 6 flip-flops. While 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.

Note that an explicit enumeration of all possible implementations is not feasible for this example: In order to show that φ^3 is not realizable with 1 flip-flop we had to enumerate and model check $1.8 \cdot 10^{19}$ different implementations which would need more than 584 years, even if one model checking run for a complete design would need only 1 ns (which is of course not a realistic assumption). Remember that we needed only 0.39 seconds for this task.

Synthesis: As an interesting side effect, if realizability can be shown, it is even 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. 6 (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 root to the 1-terminal 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 [28] to obtain a minimized circuit. Interestingly, the resulting circuit even holds additional useful properties not required by φ^3: Every time one or more requests are made, at least one of them is granted. Additionally, all requests are granted at the latest of two steps in the future (if the requests are persistent).

Experiment 5: Experiment 4 presented above already demonstrates that the exact method from Sect. 6 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. However, in Experiment 4 we did not yet make use of another essential property of the method: The method is exact even 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. This feature was not demonstrated, because the 'incomplete circuit' in this example just consists of a single Black Box.

In order to evaluate this additional feature as well, we considered a second example: This example consists of an arbiter together with

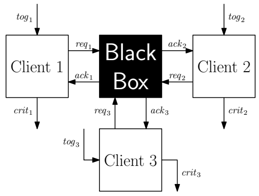


Fig. 12. Arbiter with three clients

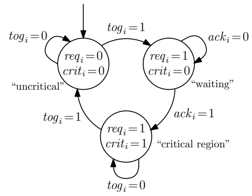


Fig. 13. Client automaton

three clients that were connected to the arbiter via request (req_i) and acknowledge (ack_i) signals as depicted in Fig. 12. Again, the arbiter was replaced by a Black Box. Here, the behavior of the clients is explicitly given and illustrated in Fig. 13: Initially, each client is in the "non-critical" state until it receives a state toggle signal (tog_i). It then changes to the "waiting" state, in which the request signal (req_i) signal is sent to the arbiter. As soon as the client receives an acknowledge (ack_i) signal from the arbiter, it moves into the "critical region" state and accordingly sets the critical ($crit_i$) signal. The client stays in the "critical region" until it receives a state toggle signal (tog_i). The toggle signal tog_i is used to model non-deterministically points in time where the client i tries to enter or leave the critical region.

We considered the property stating that no two clients are in the critical region (indicated by $crit_i = 1$) at the same time and that for each client that is not in the critical region and that receives a state toggle signal (indicated by $tog_i = 1$), there is a successor state in which the client has entered the critical region:

$$\varphi = \bigwedge_{1 \le i < j \le 3} AG\neg(crit_i \wedge crit_j)$$
$$\wedge \bigwedge_{1 \le i < j \le 3} AG\big((\neg crit_i \wedge tog_i) \to EF\, crit_i\big)$$

Using our exact method, we were able to prove that the property is realizable for this design even if the bounded memory of the Black Box is limited to only 1 flip-flop (in contrast to the previous example where we needed 2 flip-flops for an arbiter with 3 clients). The proof was completed within 171.03 minutes with a peak live BDD node count of 116678973. Automatic reordering was disabled for this experiment. For 0 internal flip-flops, we were able to prove that the property is not realizable, thus there is no combinational replacement for the Black Box such that the property is satisfied.

It is interesting to note that the realizability result for memory bounded to 1 flip-flop relies on the fact that the possible behaviors of the clients are restricted, i.e., they are defined according to Fig. 13. On the other hand, our solution does *not* assume that the arbiter has access to signals different from its input signals: The actual arbiter implementations which can be extracted from the result of the model checking run are limited to have exactly the same set of inputs as the original Black Box. Here this means that the arbiter has no inputs other than the three req_i signals and the arbiter has no possibility to 'give itself an easy time' by 'reading' internal states of the clients (which would be the case for other approaches from controller synthesis such as [8]).

While the method is able to provide interesting results as shown in this section, BDD sizes in our experiments also indicate that the exact method is applicable to benchmarks of moderate size only. This again gives us a motivation for considering *approximate* methods for solving realizability and validity questions.

8 Related Work and Discussion

Some well-known model checking tools like SMV [6] (resp. NuSMV [29]), and VIS [10] provide the definition of nondeterministic signals (see [30]–[32]) which can be used to model uncertainty. Whereas for universal subclasses of CTL like ACTL [27], abstracting parts of a design with nondeterministic signals provides sound approximations, model checking with nondeterministic signals is not able to solve realizability and validity problems for arbitrary CTL formulas – neither exactly nor approximately, see Appendix E for details. Only *universal* properties that hold in the abstraction are guaranteed to hold also in the concrete model, since these abstractions result just in simulations [33] of the concrete model. In order to preserve arbitrary CTL formulas, the stronger bi-simulations [34] would be needed which usually reduce the complexity to a much smaller extent. Therefore we work with larger approximations and compute over- and underapproximations for CTL formulas instead of standard model checking.

Symbolic Z-simulation, our simplest method to model Black Boxes based on ternary $(0, 1, X)$-logic, is related to Symbolic Trajectory Evaluation (STE) [14], [35]. In STE, some signals of the design are automatically abstracted to X based on the property under consideration. The success of STE relies on the fact that it allows only very restricted properties: Typically, properties used in STE are arguing about bounded time windows only; these properties (called 'simple assertions' in [14]) have the special form $A \Rightarrow C$ where A and C are so-called trajectory formulas. (The antecedent A expresses constraints on signals at different times t, and the consequent C expresses requirements that should hold on signals at (some other) times t'.) STE solves the model checking problem by considering symbolic representations for all runs of the system ('trajectories') fulfilling A and all sequences of signals fulfilling C. The traces fulfilling A are over-approximated using ternary $(0, 1, X)$-logic. In contrast to that, approximations used in our method are not necessarily bound to $(0, 1, X)$-logic and we support the full temporal logic CTL.

Combinational Black Boxes in incomplete designs may be seen as Uninterpreted Functions (UIFs), as the value of the outputs is unknown in both cases. UIFs have been mainly used in the verification of pipelined microprocessors [15], [36]–[38] and model functional blocks without sequential behavior. Functional consistency constraints ensure that UIFs produce the same output values when provided with same input values. The approaches mentioned above are restricted to the verification of invariants whereas our method is able to deal with the full temporal logic CTL and with sequential Black Boxes.

During the last years Bounded Model Checking (BMC) has proved to be effective for the verification of safety properties (or the more general LTL properties as presented in the original paper [39]). Initially, BMC was mainly used for bug finding (falsification of safety properties); in the meantime there are several promising approaches enabling verification as well, e.g. k-induction [40] or interpolation [41]. By reducing the search for an error path of fixed length k to a satisfiability problem, BMC profits from the power of efficient modern SAT solvers. Even without any abstraction SAT solvers are often successful by focussing their reasoning to the parts of the system which are important for the verification of the property at hand. The aspect of abstractions by using Black Boxes in our work is related to automatic abstraction techniques like localization reduction [16] in this context. Localization reduction abstracts away parts of the sequential hardware design (e.g. beginning with flip flops not occurring in the property and the logic required to compute their next state). Then a complete verification technique is used for the abstracted system (e.g. BDD based fixed point computation [42], [43], BMC with a completeness threshold syntactically determined in the abstract system [18], or BMC with k-induction [17]). If there is a counterexample in the abstract model, it is checked by BMC for the concrete model. If the abstract counterexample is not a counterexample in the concrete model, then abstraction refinement is performed (e.g. by using Counter-Example Guided Abstraction

Refinement (CEGAR) [44] (e.g. in [42]) or by analyzing the proof of nonexistence of counterexamples of a certain length (e.g. in [17], [43])). The procedure stops when a concretizable counterexample is found or when the property can be proven for the abstract model. In contrast to the SAT-based approaches mentioned above, our approach uses BDD-based model checking. It is not restricted to safety properties, but works for the full class of CTL formulas. On the other hand, our abstraction methods are not automatic, but they are based on user knowledge about the design and on user assumptions about the importance of certain parts of the design for the property to be verified. Whereas BDD-based methods are widely believed to be applicable to medium-sized problems only (because of the “state-space explosion problem”), our work shows that they may be competitive even for safety properties, if they are complemented with abstraction methods which allow property specific abstractions of different strengths and if the property allows a non-trivial amount of abstraction of the full model. Compared to SAT-based approaches using localization reduction, our approach contains the additional idea of combining stronger approximation methods with weaker ones: Whereas abstraction by Black Boxes makes the verification task easier by replacing complex parts of the overall system, the remaining system may still suffer from complexity problems due to a large number of variables, if the interfaces of some Black Boxes are wide (i.e. contain many signals). In such cases the number of variables may be reduced by a flexible application of Z-modeling.

In the context of software model checking there is a long line of research on model checking branching properties of partial models in different shapes. Modal Transition Systems (MTSs) [11], [45] exhibit *must*- and *may*-transitions between states,[7] while the state labels in MTSs always have concrete values *true* and *false*. Bruns and Godefroid [12], [46] introduced Partial Kripke Structures (PKSs) where labels are allowed to have *must* and *may* values, while the transitions between states are fixed. Kripke Modal Transition Systems (KMTSs) introduced by Huth et al. [13] allow both *must* and *may* labels and transitions. All of these modeling formalisms have been shown to be equivalent [47], [48]. Additional formalisms like Belnap Transition Systems [49] and Generalized KMTSs [50] are also basically equivalent to KMTSs [51]. Model checking for these systems is based on 3-valued logic[8] and is called 3-valued model checking.

The most prominent area of application for KMTSs (and their equivalent counterparts) is the abstraction of possibly infinite state spaces to finite state spaces by abstract interpretation of software programs (leading to must resp. may transitions between abstract states and must resp. may labels of abstract states) [52], [53]. In our work we do not combine concrete states into abstract states, but we abstract from *components* of a system. By doing so, we abstract from the (possibly complex) internal functionality of components before a symbolic representation of the overall system is computed. (I.e. we remove the restrictions to the overall behavior resulting from the concrete implementation of the component.) Moreover, abstraction of already existing components is not the only application of our methods for solving realizability and validity questions.

As in Sect. 5.2 of our paper, 3-valued model checking (e.g. according to [13]) is reduced to recursive computations of sets of states possibly and definitely fulfilling a property φ. The evaluation of negation, disjunction, and fix point iterations for EG and EU are exactly the same in [13] and in our work (see Def. 16, lemmas 10, 11). However, the evaluation of EX differs: In our work we consider only possible transitions (i.e. may transitions) for EX evaluation: If all possible transitions from a state lead to states definitely satisfying ψ, then this state definitely satisfies ψ (see second part of Def. 15 on page 7).[9] We can choose this approach, since it is guaranteed in our application that for each completion of an incomplete design there ‘remains at least one of the possible transitions’. The evaluation of EX in [13] uses must-transitions: $EX\psi$ holds definitely in a state q, if there is a must transition to another state q' definitely fulfilling ψ. This approach leads to less accurate results (consider e.g. the case that from q there is no must transition but several may transitions which all lead to states fulfilling ψ).

A main focus of our work is the question of *how* to compute possible transitions (based on transition functions δ_i) and 3-valued information on atomic propositions (based on output functions λ_i) in an efficient and symbolic manner for our application (realizability and validity questions for incomplete designs). The usage of Z-, Z/Z_i- and Z_i-simulation provides different options to trade efficiency against accuracy while computing such information.

Beyond that, accuracy can be increased by the inclusion of Z_i-variables into the state space (see Sect. 5.3) which to the best of our knowledge does not have an analogon in the context of existing work wrt. 3-valued model checking. Conceptually, this technique increases accuracy by case splitting wrt. Black Box outputs.

Apart from the standard semantics for 3-valued model checking there is also the so-called thorough semantics [46] which is more accurate: Thorough model checking provides an *exact* solution to the question whether all completions of a KMTS to a complete Kripke structure fulfill a given property. Although our simpler approximations make use of 3-valued model checking, we can not use thorough model checking to obtain more exact solutions to realizability and validity questions. The main difference lies in the notion of completion: Completions in our sense consist in replacements of Black Boxes by sequential designs, whereas completions in the context of KMTSs basically consist in replacements of may transitions by fixed transitions and replacements of unknowns for atomic propositions by concrete values (0 or 1). Of course, replacements of Black Boxes by incomplete designs lead to replacements of may transitions by fixed transitions and to replacements of unknowns for atomic propositions by concrete values as well, but not all of these replacements are possible, since *the replacements of different unknowns by concrete values are correlated by the given incomplete design*. *Arbitrary* replacements as for KMTSs are not allowed. Thus, in our application we could neither rely on negative answers of thorough model checking for validity nor on positive answers for realizability. For an *exact* solution to realizability we have to consider in addition that Black Boxes are replaced by sequential designs and thus their outputs can not take arbitrary values (e.g. identical sequences at the Black Box inputs produce identical output sequences, see example on page 10, Sect. 6.1). Altogether thorough model checking solves a completely different problem – although there are similarities in spirit wrt. increasing the accuracy of approximate methods.

In [54] the concept of 3-valued model checking was generalized to multi-valued model checking based on (multi-valued) quasi-Boolean algebras. Apart from 3-valued model checking as a special case of multi-valued model checking, this generalization allows interesting applications to other problems (by an appropriate choice of the underlying quasi-Boolean algebra), e.g. to temporal logic query checking [55], [56] and vacuity checking [57], [58]. Multi-valued model checking problems can be solved by a multi-valued symbolic model checker [54] or can be reduced to several classical model checking problems [46], [59].

7. In our paper ‘may transitions’ are called ‘possible transitions’.

8. The three values are called 0, 1, X in our paper.

9. The additional existential quantification $\exists x'$ in the second part of Def. 15 comes from subtleties due to the translation of sequential designs into Kripke structures.

Problems especially related to our exact approach from Sect. 6 were solved in symbolic controller synthesis [8] and in the context of the 'repair problem' [60] which asks whether and how an erroneous design can be repaired so that the property is satisfied. In these approaches upper bounds to the number of states of the 'Black Boxes' are assumed, too. In [8] the Black Boxes have unlimited access to all signals in the design. In contrast to that, our method takes into account that the Black Boxes are only able to read the input signals connected to them. Thus, the exact method from Sect. 6 is able to provide an exact answer even for the case that the Black Boxes do not have *global knowledge*. The approach from [60] also considers limited access to signals in the design by universally quantifying variables from a strategy which was selected for an appropriate game. However, this step may destroy the selected strategy, and thus [60] provides only a heuristical method.

Finally, a related problem is given in [61] where a Finite State Machine (FSM) interacts with *one* unknown component (Black Box). In [61] solutions of parallel 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. The approach from [61] provides an exact solution and by explicitly modeling communication channels between the FSM and the Black Box it takes into account that the Black Box may have only restricted access to the signals of the surrounding design (in contrast to controller synthesis approaches such as [8]). Since this work was done in the context of logic synthesis (and not verification), the specification is not given by a temporal CTL formula as in our work, but by another FSM.

9 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 Black Boxes with larger approximation and thus faster, if they are less relevant in terms of the CTL formula, and at the same time we do not necessarily lose important information which can only be provided by more exact methods.

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 symbolic 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 [27] 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.

Acknowledment

We are indebted to Christian Miller whose efforts preparing the experimental evaluation helped us very much.

References

[1] C. Scholl and B. Becker, "Checking Equivalence for Partial Implementations," in *Design Automation Conf.*, 2001, pp. 238–243.
[2] T. Nopper and C. Scholl, "Approximate symbolic model checking for incomplete designs," in *FMCAD*, ser. LNCS, vol. 3312. Austin, Texas: Springer Verlag, November 2004, pp. 290–305.
[3] E. M. Clarke, E. A. Emerson, and A. P. 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.
[4] R. E. Bryant, "Graph - based algorithms for Boolean function manipulation," *IEEE Trans. on CAD*, vol. 35, no. 8, pp. 677–691, 1986.
[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Information and Computation*, vol. 98(2), pp. 142–170, 1992.
[6] K. L. McMillan, *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.
[7] A. Pnueli and R. Rosner, "On the Synthesis of a Reactive Module," in *ACM Symp. on Principles of Programming Languages*. ACM Press, 1989, pp. 179–190.
[8] E. Asarin, O. Maler, and A. Pnueli, "Symbolic controller synthesis for discrete and timed systems," in *Hybrid Systems*. Springer, 1995, pp. 1–20.
[9] A. Pnueli and R. Rosner, "Distributed systems are hard to synthesize," in *IEEE Symp. on Foundations of Computer Science*, 1990, pp. 746–757.
[10] T. VIS Group, "VIS: A system for verification and synthesis," in *CAV*, ser. LNCS, vol. 1102. Springer Verlag, 1996, pp. 428–432.
[11] K. G. Larsen and B. Thomsen, "A modal process logic," in *LICS*. IEEE Computer Society, 1988, pp. 203–210.
[12] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *CAV*, ser. LNCS, vol. 1633. Springer, 1999, pp. 274–287.
[13] M. Huth, R. Jagadeesan, and D. Schmidt, "Modal transition systems: A foundation for three-valued program analysis," in *European Symp. on Programming*, vol. 2028. Springer, April 2001, pp. 155+.
[14] C.-J. H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," *Formal Methods in System Design*, vol. 6, no. 2, pp. 147–189, March 1995.
[15] J. R. Burch and D. L. Dill, "Automatic verification of microprocessor control," in *Int'l Conf. on CAV*, ser. LNCS, vol. 818. Springer Verlag, 1994, pp. 68–80.
[16] R. Kurshan, *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, 1994.
[17] B. Li, C. Wang, and F. Somenzi, "Abstraction refinement in symbolic model checking using satisfiability as the only decision procedure," *STTT*, vol. 7, no. 2, pp. 143–155, 2005.
[18] D. Kroening, "Computing over-approximations with bounded model checking," *Electr. Notes Theor. Comput. Sci.*, vol. 144, no. 1, pp. 79–92, 2006.
[19] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary decision diagrams," *ACM Computing Surveys*, vol. 24, pp. 293–318, 1992.
[20] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.
[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] T. Filkorn, "Functional extension of symbolic model checking," in *Int'l Conf. on CAV*. Springer, 1992, pp. 225–232.
[23] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking," in *Int'l Conf. on CAV*, ser. LNCS, vol. 1855. Springer Verlag, 2000, pp. 124–138.
[24] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.
[25] 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.
[26] R. Hojati, S. C. Krishnan, and R. K. Brayton, "Early quantification and partitioned transition relations," in *IEEE Int'l Conf. on Computer Design*, 1996, pp. 12–19.
[27] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.
[28] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," University of Berkeley, Tech. Rep., 1992.
[29] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: A new symbolic model verifier," in *Int'l Conf. on CAV*. Springer Verlag, July 1999, pp. 495–499.
[30] K. L. McMillan, *The SMV language*, Cadence Berkeley Labs, 1999.
[31] ——, *The SMV system - for SMV version 2.5.4*, Carnegie Mellon University, 2000.
[32] T. Villa, G. Swamy, and T. Shiple, *VIS User's Manual*, Electronics Research Laboratory, University of Colorado at Boulder, 1996.
[33] R. Milner, "An algebraic definition of simulation between programs," in *Proceedings of the 2nd international joint conference on Artificial*

intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1971, pp. 481–489.

[34] D. Park, "Concurrency and automata on infinite sequences," in *Proceedings of the 5th GI-Conference on Theoretical Computer Science*. London, UK: Springer-Verlag, 1981, pp. 167–183.

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

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

[37] R. E. Bryant, S. German, and M. N. 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.

[38] S. K. Lahiri, S. A. Seshia, and R. E. Bryant, "Modeling and verification of out-of-order microprocessors in UCLID," in *FMCAD*, ser. LNCS, vol. 2517. Springer-Verlag, 2002, pp. 142–159.

[39] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 1579. Springer Verlag, 1999.

[40] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties Using Induction and a SAT-solver," in *Int'l Conf. on Formal Methods in CAD*, ser. LNCS, W. A. Hunt Jr. and S. D. Johnson, Eds., vol. 1954. Springer, 2000, pp. 407–420.

[41] K. L. McMillan, "Interpolation and SAT-Based Model Checking," in *Int'l Conf. on CAV*, ser. LNCS, W. A. Hunt Jr. and F. Somenzi, Eds. Springer, 2003.

[42] D. Wang, P.-H. Ho, J. Long, J. Kukula, Y. Zhu, T. Ma, and R. Damiano, "Formal property verification by abstraction refinement with formal, simulation and hybrid engines," in *DAC*. ACM, 2001, pp. 35–40.

[43] C. Wang, B. Li, H. Jin, G. D. Hachtel, and F. Somenzi, "Improving ariadne's bundle by following multiple threads in abstraction refinement," in *ICCAD*. IEEE Computer Society / ACM, 2003, pp. 408–415.

[44] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," in *Int'l Conf. on CAV*, ser. LNCS, vol. 1855. Springer, 2000, pp. 154–169.

[45] K. G. Larsen and L. Xinxin, "Equation solving using modal transition systems," in *LICS*. IEEE Computer Society, 1990, pp. 108–117.

[46] G. Bruns and P. Godefroid, "Generalized Model Checking: Reasoning about Partial State Spaces," in *Int'l Conf. on Concurrency Theory*, vol. 1877. Springer, 2000, pp. 168–182.

[47] P. Godefroid and R. Jagadeesan, "On the expressiveness of 3-valued models," in *VMCAI*, ser. Lecture Notes in Computer Science, L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, Eds., vol. 2575. Springer, 2003, pp. 206–222.

[48] A. Gurfinkel and M. Chechik, "How thorough is thorough enough?" in *CHARME*, ser. LNCS, vol. 3725. Springer, 2005, pp. 65–80.

[49] A. Gurfinkel, O. Wei, and M. Chechik, "Systematic construction of abstractions for model-checking," in *VMCAI*, ser. Lecture Notes in Computer Science, E. A. Emerson and K. S. Namjoshi, Eds., vol. 3855. Springer, 2006, pp. 381–397.

[50] S. Shoham and O. Grumberg, "Monotonic abstraction-refinement for ctl," in *TACAS*, ser. LNCS, vol. 2988, 2004, pp. 546–560.

[51] O. Wei, A. Gurfinkel, and M. Chechik, "Mixed transition systems revisited," in *VMCAI*, ser. LNCS, vol. 5403. Springer, 2009, pp. 349–365.

[52] D. Dams, R. Gerth, and O. Grumberg, "Abstract interpretation of reactive systems," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 253–291, 1997.

[53] P. Godefroid, M. Huth, and R. Jagadeesan, "Abstraction-based model checking using modal transition systems," in *CONCUR*, ser. LNCS, vol. 2154. Springer, 2001, pp. 426–440.

[54] M. Chechik, B. Devereux, S. M. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 4, pp. 371–408, 2003.

[55] W. Chan, "Temporal-logic queries," in *Int'l Conf. on CAV*, ser. LNCS, vol. 1855. Springer, 2000, pp. 450–463.

[56] A. Gurfinkel, M. Chechik, and B. Devereux, "Temporal logic query checking: A tool for model exploration," *IEEE Trans. Software Eng.*, vol. 29, no. 10, pp. 898–914, 2003.

[57] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh, "Efficient detection of vacuity in temporal model checking," *Form. Methods Syst. Des.*, vol. 18, no. 2, pp. 141–163, 2001.

[58] A. Gurfinkel and M. Chechik, "How vacuous is vacuous?" in *TACAS*, ser. LNCS, vol. 2988, 2004, pp. 451–466.

[59] ——, "Multi-valued model checking via classical model checking," in *CONCUR*, ser. Lecture Notes in Computer Science, R. M. Amadio and D. Lugiez, Eds., vol. 2761. Springer, 2003, pp. 263–277.

[60] B. Jobstmann, A. Griesmayer, and R. Bloem, "Program repair as a game," in *CAV*, ser. LNCS, vol. 3576, 2005, pp. 226–238.

[61] N. Yevtushenko, T. Villa, R. K. Brayton, A. Petrenko, and A. L. Sangiovanni-Vincentelli, "Solution of parallel language equations for logic synthesis," in *IEEE Int'l Conf. on Computer-Aided Design*. IEEE Press, 2001, pp. 103–110.

[62] A. Biere and R. Brummayer, "Consistency checking of all different constraints over bit-vectors within a sat solver," in *FMCAD*. IEEE, 2008, pp. 1–4.

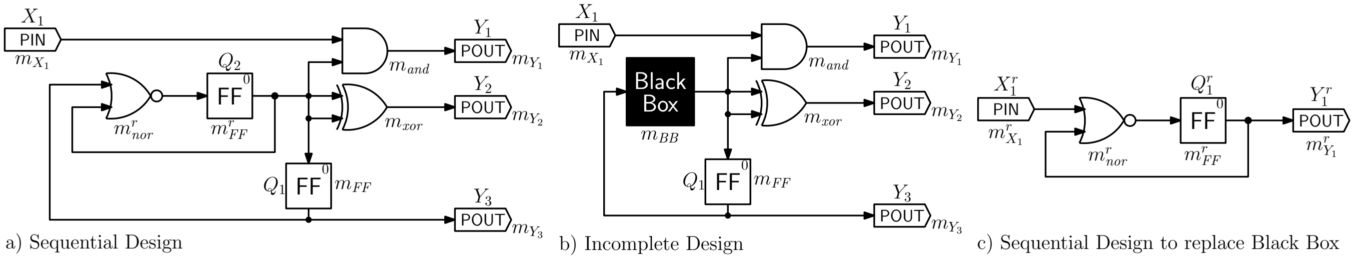
Tobias Nopper studied computer science from 1996 to 2003 at the University of Freiburg, Germany, where he also received the Dipl.-Inform. degree in 2003. From 2004 to 2010, he was with the DFG transregional collaborative research center "AVACS - Automatic Verification and Analysis of Complex Systems" at the University of Freiburg in close collaboration both with the University of Saarland and the University of Oldenburg. His research interests are formal verification of incomplete designs (circuits containing "Black Boxes"), model checking and counterexample generation.

Christoph Scholl received the Dipl.-Inform. and the Dr.-Ing. degrees in computer science from University of Saarland, Germany, in 1993 and 1997, respectively. In 2002 he received the venia legendi from University of Freiburg, Germany. In 2002/2003 he was an associate professor for computer engineering at the University of Heidelberg and in 2003 he joined the University of Freiburg as an associate professor in the Department of Computer Science. His research interests include logic synthesis, real-time operating systems, and the verification as well of digital circuits and systems and of hybrid systems. In this context a main focus of his work lies on the development of efficient (symbolic) data structures and algorithms.

APPENDIX

A. Complete and Incomplete Designs

We first give a formal definition of (complete) sequential designs and their semantics expressed by transition functions and output functions. Then we extend the formal definition to incomplete designs.



a) Sequential Design b) Incomplete Design c) Sequential Design to replace Black Box

Fig. 14. Complete and incomplete sequential designs

Complete Designs

Definition 25 (Sequential Design)**.** *Let $\mathcal{B}_{n,m} = \{f : \mathbb{B}^n \to \mathbb{B}^m\}$ be the set of boolean functions with n inputs and m outputs and let $LIB \subseteq \bigcup_{n,m\in\mathbb{N}} \mathcal{B}_{n,m}$ be a library of boolean functions. A sequential design over LIB is a tuple $D = \big(M, in, out, type, src, \vec{X}, \vec{Y}, \vec{Q}, \vec{q}^{\,0}\big)$ with the following properties:*

1) *M is a set of nodes.*
2) *$in\colon M \to \mathbb{N}$ is a function returning the number of inputs of a node.*
3) *$out\colon M \to \mathbb{N}$ is a function returning the number of outputs of a node.*
4) *$type\colon M \to LIB \cup \{PIN, POUT, FF\}$ is a function which assigns a type to each node. Nodes with type PIN model inputs of the sequential design, nodes with type $POUT$ model outputs, nodes with type FF model memory elements (flip-flops) storing single bits, and nodes with a type from LIB model gates implementing boolean functions. If $type(m) \in LIB$, then $type(m) \in \mathcal{B}_{in(m),out(m)}$, if $type(m) = PIN$, then $in(m) = 0$, $out(m) = 1$, if $type(m) \in POUT$, then $in(m) = 1$, $out(m) = 0$, and if $type(m) = FF$, then $in(m) = out(m) = 1$.*
5) *Let $M\_IN := \{(m,i) \,|\, m \in M,\ 1 \le i \le in(m)\}$ and $M\_OUT := \{(m,i) \,|\, m \in M,\ 1 \le i \le out(m)\}$. The "source" function $src\colon M\_IN \to M\_OUT$ models the connections between the nodes. $src(m,i) = (m', j)$ if the j-th output of node m' is the source for the value at the i-th input of node m.*
6) *$\vec{X} = (X_1, X_2, \ldots, X_{|\vec{X}|}) \in M^{|\vec{X}|}$ is a list of all nodes with type PIN.*
7) *$\vec{Y} = (Y_1, Y_2, \ldots, Y_{|\vec{Y}|}) \in M^{|\vec{Y}|}$ is a list of all nodes with type $POUT$.*
8) *$\vec{Q} = (Q_1, Q_2, \ldots, Q_{|\vec{Q}|}) \in M^{|\vec{Q}|}$ is a list of all nodes with type FF.*
9) *$\vec{q}^{\,0} \in \mathbb{B}^{|\vec{Q}|}$ is the initial state of D, i.e., $\forall 1 \le i \le |\vec{Q}|$ flip-flop Q_i is initialized with q^0_i.*
10) *The sequential design is free of combinational cycles, i.e., if there is a list of nodes which are connected such that they build a cycle, then there is at least one flip-flop in the cycle. More precisely, if there is $(m_1, \ldots, m_n) \in M^n$ with $\exists j,k \in \mathbb{N}\colon src(m_1, j) = (m_n, k)$ and $\forall 2 \le i \le n\colon \exists j,k \in \mathbb{N}\colon src(m_i, j) = (m_{i-1}, k)$, then there is at least one $i \in \{1, \ldots, n\}$ with $type(m_i) = FF$.*

Example. Fig. 14 a) shows an example for a sequential design with one input, three outputs, two flip-flops, and three gates implementing boolean functions nor_2, and_2, xor_2, respectively. The "source" function src is depicted by arrows: There is an arrow from the j-th output of node m' to the i-th input of node m iff $src(m,i) = (m', j)$.

Now we define the boolean functions which are computed by the nodes in a sequential design. These boolean functions are needed later on to define transition and output functions of the design.

Definition 26. *Let $D = \big(M, in, out, type, src, \vec{X}, \vec{Y}, \vec{Q}, \vec{q}^{\,0}\big)$ be a sequential design. The i-th output of a node $m \in M$ computes a boolean function $f(m,i)\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}$. In the definition of $f(m,i)$ we denote the projection function which maps $(x_1, \ldots, x_{|\vec{X}|}, q_1, \ldots, q_{|\vec{Q}|}) \in \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|}$ to x_j (q_k) simply by x_j (q_k). $f(m,i)$ is defined as*

$$f(m,i) := \begin{cases} x_j & \text{if } m = X_j \\ q_k & \text{if } m = Q_k \\ g_i(f(src(m,1)), \ldots, & \text{if } type(m) = g \text{ and } g_i \text{ is the} \\ \quad f(src(m, in(m)))) & i\text{-th output function of } g \end{cases}$$

$f(m,i)$ as defined above is well-defined, since the sequential design D does not contain any combinational cycle.

Definition 27 (Transition and Output Function)**.** *Given a sequential design $D = \big(M, in, out, type, src, \vec{X}, \vec{Y}, \vec{Q}, \vec{q}^{\,0}\big)$, the function $\vec{\delta}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Q}|}$ with $\vec{\delta} := (f(src(Q_1, 1)), \ldots, f(src(Q_{|\vec{Q}|}, 1)))$ is the function that computes the next state of the flip-flops and is thus called the* transition *function. The function $\vec{\lambda}\colon \mathbb{B}^{|\vec{Q}|} \times \mathbb{B}^{|\vec{X}|} \to \mathbb{B}^{|\vec{Y}|}$ with $\vec{\lambda} := (f(src(Y_1, 1)), \ldots, f(src(Y_{|\vec{Y}|}, 1)))$ is the function that computes the current output values and is called the* output *function.*

Incomplete Designs

We now provide a formal definition for incomplete designs that are essentially sequential designs with additional "Black Box" (BB) nodes with unknown (sequential) behavior.

Definition 28 (Incomplete Design)**.** *An incomplete design over $LIB \subseteq \bigcup_{n,m\in\mathbb{N}} \mathcal{B}_{n,m}$ is a tuple $D = \big(M, in, out, type, src, \vec{X}, \vec{Y}, \vec{Q}, \vec{q}^{\,0}\big)$ with the same properties 1)–3) and 5)–10) as in Def. 25 for sequential designs. 4) is replaced by 4') as follows:*

4') *$type\colon M \to LIB \cup \{PIN, POUT, FF, BB\}$ is a function which assigns a type to each node. Nodes with type BB model 'Black Boxes' of the incomplete design. If $type(m) \in LIB$, then $type(m) \in \mathcal{B}_{in(m),out(m)}$, if $type(m) = PIN$, then $in(m) = 0$, $out(m) = 1$, if $type(m) = POUT$, then $in(m) = 1$, $out(m) = 0$, and if $type(m) = FF$, then $in(m) = out(m) = 1$.*

Example. Figure 14 b) illustrates an incomplete design with one input, three outputs, one flip-flop, two gates implementing the boolean and_2 resp. the boolean xor_2 function and one Black Box. Just as in the case of complete sequential designs the "source" function src is depicted by arrows. More precisely, $D = \big(M, in, out, type, src, X, Y, Q, \vec{q}^{\,0}\big)$ with

1) $M = \{m_{X_1}, m_{BB}, m_{FF}, m_{and}, m_{xor}, m_{Y_1}, m_{Y_2}, m_{Y_3}\}$
2) $in\colon \{m_{BB}, m_{FF}, m_{Y_1}, m_{Y_2}, m_{Y_3}\} \mapsto 1$,
$in\colon \{m_{and}, m_{xor}\} \mapsto 2$, $in\colon \{m_{X_1}\} \mapsto 0$
3) $out\colon \{m_{BB}, m_{FF}, m_{and}, m_{xor}\} \mapsto 1$,
$out\colon \{m_{Y_1}, m_{Y_2}, m_{Y_3}\} \mapsto 0$
4) $type(m_{X_1}) = PIN$, $type(m_{BB}) = BB$, $type(m_{FF}) = FF$, $type(m_{and}) = and_2$, $type(m_{xor}) = xor_2$, $type(m_{Y_1}) = type(m_{Y_2}) = type(m_{Y_3}) = POUT$

5) $src\colon \{(m_{and},1)\} \mapsto (m_{X_1},1)$
$src\colon \{(m_{and},2),(m_{xor},1),(m_{xor},2),(m_{FF},1)\} \mapsto (m_{BB},1)$
$src\colon (m_{BB},1) \mapsto (m_{FF},1)$
$src\colon (m_{Y_1},1) \mapsto (m_{and},1)$
$src\colon (m_{Y_2},1) \mapsto (m_{xor},1)$
$src\colon (m_{Y_3},1) \mapsto (m_{FF},1)$

Replacing Black Boxes in an Incomplete Design

A Black Box m_{BB} in an incomplete design D can be replaced by any sequential design D^r (without Black Boxes and with an arbitrary number of flip-flops), as long as D^r has the same number of inputs and outputs as the Black Box. The inputs and outputs of D^r are then connected to the inputs and outputs of the former Black Box m_{BB} in D; the result of this substitution is another (possibly incomplete) sequential design $D^c = \big(M^c, in^c, out^c, type^c, src^c, \vec{X}^c, \vec{Y}^c, \vec{Q}^c, \vec{q}^{0c}\big)$. The exact definition of a substitution of a Black Box by a sequential design is as follows:

Definition 29 (Replacement of a Black Box in an Incomplete Design)**.** *Let $D = \big(M, in, out, type, src, \vec{X}, \vec{Y}, \vec{Q}, \vec{q}^0\big)$ be an incomplete design and let $m_{BB} \in M$ be a Black Box in D ($type(m_{BB}) = BB$). The replacement of m_{BB} by a sequential design $D^r = \big(M^r, in^r, out^r, type^r, src^r, \vec{X}^r, \vec{Y}^r, \vec{Q}^r, \vec{q}^{0r}\big)$ with $\big|\vec{X}^r\big| = in(m_{BB})$ and $\big|\vec{Y}^r\big| = out(m_{BB})$ is defined as an incomplete design $D^c = \big(M^c, in^c, out^c, type^c, src^c, \vec{X}^c, \vec{Y}^c, \vec{Q}^c, \vec{q}^{0c}\big)$ as follows:*

- $M^c := \big(M \setminus \{m_{BB}\}\big) \cup \big(M^r \setminus \{m' \in M^r \,\big|\, type(m') \in \{PIN, POUT\}\}\big)$
- $src^c(m,i) := \begin{cases} src(m,i) & \textit{if } m \in M \wedge \forall j\ src(m,i) \neq (m_{BB}, j) \\ src^r(Y^r_j, 1) & \textit{if } m \in M \wedge src(m,i) = (m_{BB}, j) \\ src^r(m,i) & \textit{if } m \in M^r \wedge \forall j\ src^r(m,i) \neq (X^r_j, 1) \\ src(m_{BB}, j) & \textit{if } m \in M^r \wedge src^r(m,i) = (X^r_j, 1) \end{cases}$
- $type^c(m) := \begin{cases} type(m), & \textit{if } m \in M \\ type^r(m), & \textit{if } m \in M^r \end{cases}$
- $in^c(m) := \begin{cases} in(m), & \textit{if } m \in M \\ in^r(m), & \textit{if } m \in M^r \end{cases}$
- $out^c(m) := \begin{cases} out(m), & \textit{if } m \in M \\ out^r(m), & \textit{if } m \in M^r \end{cases}$
- $\vec{X}^c := \vec{X}$, $\vec{Y}^c := \vec{Y}$, $\vec{Q}^c := (Q_1, \ldots, Q_{|\vec{Q}|}, Q^r_1, \ldots, Q^r_{|\vec{Q}^r|})$
- $\vec{q}^{0c} := (q^0_1, \ldots, q^0_{|\vec{Q}|}, q^{0r}_1, \ldots, q^{0r}_{|\vec{Q}^r|})$

Example. The sequential design in Fig. 14 a) results from the incomplete design in Fig. 14 b) by replacing the Black Box with the sequential design in Fig. 14 c).

B. Proof of Lemma 2

In the proof we make use of the notions defined in Appendix A for complete and incomplete sequential designs.

Proof: The proof is by induction on the structure of D. Of course, the statement is true for $type(m) \in \{PIN, FF, BB\}$. For $type(m) \in \{and_2, or_2\}$ the proof follows easily from $f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}}, f_Z(src(m,2))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} \in \{0,1,Z\}$ by the induction hypothesis. For $type(m) = not$ we have

$$(\overline{f_Z(src(m,1))}|_{Z\leftarrow\overline{Z}})|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = (\overline{f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}}})|_{Z\leftarrow\overline{Z}}$$

which is also in $\{0,1,Z\}$, if we assume $f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} \in \{0,1,Z\}$ by induction hypothesis. □

C. Proof of Lemma 3

In the proof we make use of the notions defined in Appendix A for complete and incomplete sequential designs.

Proof: The lemma is proved by induction on the structure of D. For $type(m) \in \{PIN, FF\}$ $f_Z(m,j) = f_{Z_i}(m,j)$ and thus the statement holds. For $type(m) = BB$, the precondition of the lemma never holds and nothing has to be proved.

Now consider $type(m) = and_2$. Then

$$\begin{aligned} f_Z(m,1)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} &= \Big(f_Z(src(m,1)) \cdot f_Z(src(m,2))\Big)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} \\ &= \Big(f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}}\Big) \cdot \Big(f_Z(src(m,2))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}}\Big). \end{aligned}$$

- Case 1: $f_Z(m,1)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 1$. Then
$$f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = f_Z(src(m,2))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 1$$
and by induction hypothesis
$$f_{Z_i}(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = f_{Z_i}(src(m,2))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 1.$$
Thus, $f_{Z_i}(m,1)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 1$.
- Case 2: $f_Z(m,1)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 0$. Since $f_Z(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}}$ and $f_Z(src(m,2))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} \in \{0,1,Z\}$ because of Lemma 2, one of these two terms has to be 0, assume w.l.o.g. the first one. Then $f_{Z_i}(src(m,1))|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 0$ by induction hypothesis and $f_{Z_i}(m,1)|_{\substack{\vec{q}=\vec{\beta}\\ \vec{x}=\vec{\gamma}}} = 0$.

The cases $type(m) = or_2$ and $type(m) = not$ can be proved in an analogous manner and are omitted here. □

D. Proof of Theorem 16

Proof: The theorem is proved by induction on the structure of the CTL property φ. For cases $\varphi = x_i$, $\varphi = y_i$, $\varphi = \neg\psi$ and $\varphi = \psi_1 \vee \psi_2$, this is trivial to show, since for these cases, $\chi_{Sat_A^{\mathsf{appr,incl}}(\cdot)}$ ($\chi_{Sat_E^{\mathsf{appr,incl}}(\cdot)}$) and $\chi_{Sat_A^{\mathsf{func}}(\cdot)}$ ($\chi_{Sat_E^{\mathsf{func}}(\cdot)}$) are defined in the same way (see Def. 23). The proofs for $EG\psi$ and $E\varphi_1 U \varphi_2$ follow from the proof for $EX\psi$.

Before we show that $\chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)} = \chi_{Sat_A^{\mathsf{func}}(EX\psi)}$ and $\chi_{Sat_E^{\mathsf{appr,incl}}(EX\psi)} = \chi_{Sat_E^{\mathsf{func}}(EX\psi)}$, we first state a few facts necessary for the proof.

Facts. *Let $f\colon \mathbb{B}^{n+1} \to \mathbb{B}$ be a boolean function over variables $(x_1, \ldots, x_n, Z)$ resulting from symbolic Z-simulation according to Def. 9. Let y be a boolean variable. Then:*

It follows from Lemma 2 that f is monotonically increasing in Z, i.e., $f|_{Z=0} \le f|_{Z=1}$. (12)

$$\forall Z f = f|_{Z=0} \cdot f|_{Z=1} \overset{(12)}{=} f|_{Z=0} \tag{13}$$

$$\exists Z f = f|_{Z=0} + f|_{Z=1} \overset{(12)}{=} f|_{Z=1} \tag{14}$$

$$f|_{Z=0} \cdot \overline{f}|_{Z=1} = 0, \textit{ since } f|_{Z=0} \cdot \overline{f}|_{Z=1} \overset{(12)}{\le} f|_{Z=1} \cdot \overline{f}|_{Z=1} = 0 \tag{15}$$

$$\begin{aligned} \big(\exists Z(f \equiv y)\big) &= \exists Z(\overline{f} \cdot \overline{y} + f \cdot y) \\ &= (\overline{f} \cdot \overline{y} + f \cdot y)|_{Z=0} + (\overline{f} \cdot \overline{y} + f \cdot y)|_{Z=1} \\ &= (\overline{f}|_{Z=0} + \overline{f}|_{Z=1}) \cdot \overline{y} + (f|_{Z=0} + f|_{Z=1}) \cdot y \\ &\overset{(13),(14)}{=} \overline{f}|_{Z=0} \cdot \overline{y} + f|_{Z=1} \cdot y \end{aligned} \tag{16}$$

$$\begin{aligned} \overline{\big(\exists Z(f \equiv y)\big)} &\overset{(16)}{=} \overline{(\overline{f}|_{Z=0} \cdot \overline{y} + f|_{Z=1} \cdot y)} \\ &= f|_{Z=0} \cdot \overline{f}|_{Z=1} + f|_{Z=0} \cdot \overline{y} + \overline{f}|_{Z=1} \cdot y + y \cdot \overline{y} \\ &\overset{(15)}{=} f|_{Z=0} \cdot \overline{y} + \overline{f}|_{Z=1} \cdot y \end{aligned} \tag{17}$$

Additionally, it follows directly from Def. 22 that for functions $f, g_1, \ldots, g_n\colon \mathbb{B}^{n+1} \to \mathbb{B}$ over variables $(\vec{x}, Z)$ that are monotonically increasing in Z, $f|^{cZ,\mathsf{impr}}_{\vec{x}\leftarrow\vec{g}}$ is also monotonically increasing in Z. (7)

As defined in Defs. 18, 19 and 23 (with $n = |\vec{q}|$):

$\chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)}(\vec{q}, \vec{x}, \vec{Z}_o)$

$$= \forall \vec{q}'\Big(\Big(\exists \vec{Z}_l \prod_{i=1}^{n} \big(\exists Z(\delta_i \equiv q_i')\big)\Big) \rightarrow \big(\exists \vec{x}\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}\big)|_{\vec{q}\leftarrow\vec{q}'}\Big)$$

$$= \forall \vec{Z}_l \forall \vec{q}'\Big(\Big(\prod_{i=1}^{n} \big(\exists Z(\delta_i \equiv q_i')\big)\Big) \rightarrow \big(\exists \vec{x}\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}\big)|_{\vec{q}\leftarrow\vec{q}'}\Big)$$

$$\chi_{Sat_A^{\mathsf{func}}(EX\psi)}(\vec{q},\vec{x},\vec{Z}_o) = \forall \vec{Z}_l \forall Z\big(\big(\exists \vec{x}\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{func}}(\psi)}\big)|_{\vec{q}\leftarrow\vec{\delta}}^{cZ,\mathsf{impr}}\big)$$

We now define $f := \big(\exists \vec{x}\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{appr,incl}}(\psi)}\big) \stackrel{\text{IA}}{=} \big(\exists \vec{x}\forall \vec{Z}_o \chi_{Sat_A^{\mathsf{func}}(\psi)}\big)$. ('IA' means 'by induction assumption'.) Note that f does only depend on variables $\vec{q}$. Then we only have to show that

$$\forall \vec{q}'\Big(\prod_{i=1}^{n} \big(\exists Z(\delta_i \equiv q_i')\big) \rightarrow f|_{\vec{q}\leftarrow\vec{q}'}\Big) = \forall Z\left(f|_{\vec{q}\leftarrow\vec{\delta}}^{cZ,\mathsf{impr}}\right)$$

to conclude $\chi_{Sat_A^{\mathsf{appr,incl}}(EX\psi)} = \chi_{Sat_A^{\mathsf{func}}(EX\psi)}$. This is proved by induction on the number of state bits $n = |\vec{q}|$:

- $n = 0$, i.e., no composition is needed:

$$\forall \vec{q}'\Big(\prod_{i=1}^{n} \big(\exists Z(\delta_i \equiv q_i)\big) \rightarrow f|_{\vec{q}\leftarrow\vec{q}'}\Big)$$
$$= 1 \rightarrow f|_{\vec{q}\leftarrow\vec{q}'} = f = \forall Z f = \forall Z\Big(f|_{\vec{q}\leftarrow\vec{\delta}}^{cZ,\mathsf{impr}}\Big)$$

- $n-1 \rightarrow n$: Let $\vec{q}^{-1} := (q_1,\ldots,q_{n-1})$, $\vec{q}'^{-1} := (q_1',\ldots,q_{n-1}')$ and $\vec{\delta}^{-1} := (\delta_1,\ldots,\delta_{n-1})$.

$$
\begin{aligned}
& \forall \vec{q}'\Big(\prod_{i=1}^{n} \big(\exists Z(\delta_i \equiv q_i')\big) \rightarrow f|_{\vec{q}\leftarrow\vec{q}'}\Big) \\
= \;& \forall \vec{q}'\Big(\overline{\big(\exists Z(\delta_n \equiv q_n')\big)} + \Big(\prod_{i=1}^{n-1} \big(\exists Z(\delta_i \equiv q_i')\big) \rightarrow f|_{\vec{q}\leftarrow\vec{q}'}\Big)\Big) \\
= \;& \forall q_n'\Big(\overline{\big(\exists Z(\delta_n \equiv q_n')\big)} + \\
& \qquad \forall \vec{q}'^{-1}\Big(\prod_{i=1}^{n-1} \big(\exists Z(\delta_i \equiv q_i')\big) \rightarrow \big(f|_{q_n\leftarrow q_n'}\big)|_{\vec{q}^{-1}\leftarrow\vec{q}'^{-1}}\Big)\Big) \\
\stackrel{\text{IA}}{=} \;& \forall q_n'\Big(\overline{\big(\exists Z(\delta_n \equiv q_n')\big)} + \forall Z\Big(\big(f|_{q_n\leftarrow q_n'}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)\Big) \\
\stackrel{(17),(7),(13)}{=} \;& \forall q_n'\Big(\overline{\delta}_n|_{Z=1}\cdot q_n' + \delta_n|_{Z=0}\cdot \overline{q_n'} \\
& \qquad +\Big(\big(f|_{q_n\leftarrow q_n'}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0}\Big) \\
= \;& \Big(\delta_n|_{Z=0} + \Big(\big(f|_{q_n=0}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0}\Big) \\
& \cdot \Big(\overline{\delta}_n|_{Z=1} + \Big(\big(f|_{q_n=1}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0}\Big) \\
= \;& \delta_n|_{Z=0}\cdot\overline{\delta}_n|_{Z=1} + \delta_n|_{Z=0}\cdot\Big(\big(f|_{q_n=1}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0} \\
& + \overline{\delta}_n|_{Z=1}\cdot\Big(\big(f|_{q_n=0}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0} \\
& + \Big(\big(f|_{q_n=0}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0}\cdot\Big(\big(f|_{q_n=1}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)|_{Z=0} \\
\stackrel{(15)}{=} \;& \Big(\overline{\delta}_n|_{Z\leftarrow\overline{Z}}\cdot\Big(\big(f|_{q_n=0}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big) + \delta_n\cdot\Big(\big(f|_{q_n=1}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big) \\
& \quad + \Big(\big(f|_{q_n=0}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)\cdot\Big(\big(f|_{q_n=1}\big)|_{\vec{q}^{-1}\leftarrow\vec{\delta}^{-1}}^{cZ,\mathsf{impr}}\Big)\Big)|_{Z=0} \\
\stackrel{\text{Def. }22}{=} \;& \Big(f|_{\vec{q}\leftarrow\vec{\delta}}^{cZ,\mathsf{impr}}\Big)|_{Z=0} \stackrel{(7),(13)}{=} \forall Z\Big(f|_{\vec{q}\leftarrow\vec{\delta}}^{cZ,\mathsf{impr}}\Big)
\end{aligned}
$$

$\chi_{Sat_E^{\mathsf{appr,incl}}(EX\psi)} = \chi_{Sat_E^{\mathsf{func}}(EX\psi)}$ can be shown analogously. □

E. 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 [30]–[32]. Nondeterministic signals can be handled exactly as primary inputs, leading to a standard CTL model checking procedure for designs containing nondeterministic signals. Since the functionality of Black Boxes is not known, one could assume that nondeterministic signals for handling Black Box outputs would provide at least approximate solutions to the realizability or validity problem.

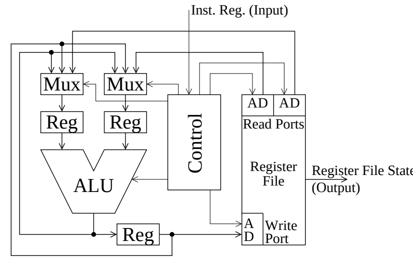
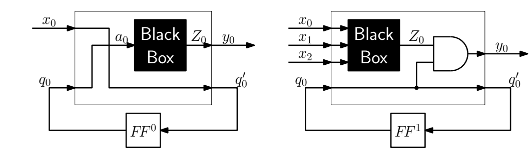


Fig. 15. Pipelined ALU



a) First counterexample   b) Second counterexample

Fig. 16. Counterexamples

As a first example for this, we consider the well-known pipelined ALU circuit from [5] (see Fig. 15). In [5], 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; $\mathbf{R}_{i,j}$ is the value of the j-th Bit of the i-th register in the register file):[10]

$$AG \bigwedge_{i,j} \Big(\big(EXEX\ \mathbf{R}_{i,j}\big) \equiv \big(AXAX\ \mathbf{R}_{i,j}\big)\Big) \tag{18}$$

$$AG \bigwedge_{i,j} \Big(\big(EXEXEX\ \mathbf{R}_{i,j}\big) \equiv \big(AXAXAX\ \mathbf{R}_{i,j}\big)\Big) \tag{19}$$

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 (19) is not satisfied.[11] 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 (18) and formula (19).

Although the previous example already shows that the usage of nondeterministic signals leads to non-exact results, we will have a closer look at two small examples to show that there is no interpretation of the results as some kind of approximation to the solution. (Here we consider SMV, but similar results hold for VIS as well.)

10. $(AX)^2$ is short for $AXAX$ and $(AX)^3$ is short for $AXAXAX$; same for $(EX)^i$.

11. Using VIS, the verification already fails for formula (18) — this is due to a slightly different modeling of automata by Kripke structures in VIS and SMV.

	$AG \bigwedge_{i,j} \big((EXEX\ \mathbf{R}_{i,j}) \equiv (AXAX\ \mathbf{R}_{i,j})\big)$										$AG \bigwedge_{i,j} \big((EXEXEX\ \mathbf{R}_{i,j}) \equiv (AXAXAX\ \mathbf{R}_{i,j})\big)$									
	Relational preimage				Functional preimage				VIS (non-det.)		Relational preimage				Functional preimage				VIS (non-det.)	
word width	BDD nodes	memory used	time	result	BDD nodes	memory used	time	result	time	result	BDD nodes	memory used	time	result	BDD nodes	memory used	time	result	time	result
2	67K	5975K	0.37	valid	7K	4912K	0.05	valid	0.3	failed	129K	7024K	0.39	unknown	8176	4928K	0.05	unknown	0.4	failed
4	242K	9073K	0.98	valid	9K	4977K	0.11	valid	1.1	failed	267K	9509K	1.16	unknown	16K	5097K	0.15	unknown	1.1	failed
6	89K	6467K	1.86	valid	12K	5070K	0.19	valid	5.0	failed	222K	8737K	1.91	unknown	25K	5284K	0.24	unknown	5.1	failed
8	152K	7628K	2.80	valid	17K	5195K	0.30	valid	98.1	failed	318K	15M	3.06	unknown	32K	5444K	0.37	unknown	111.8	failed
12	430K	17M	7.01	valid	30K	5473K	0.60	valid	32.6	failed	590K	28M	8.02	unknown	47K	5790K	0.75	unknown	35.0	failed
16	624K	29M	11.64	valid	47K	5829K	1.03	valid	88.3	failed	824K	32M	12.61	unknown	64K	6153K	1.32	unknown	97.2	failed
24	1038K	36M	26.15	valid	55K	6123K	2.43	valid	386.4	failed	1416K	43M	29.39	unknown	94K	11M	2.98	unknown	402.5	failed
32	881K	34M	29.36	valid	127K	12M	4.07	valid	TO	failed	1541K	45M	31.83	unknown	127K	12M	4.98	unknown	TO	failed
48	1566K	46M	48.35	valid	214K	13M	9.71	valid	TO	failed	1542K	46M	59.41	unknown	214K	22M	11.67	unknown	TO	failed
64	1098K	41M	68.33	valid	409K	25M	17.63	valid	TO	failed	1089K	41M	76.83	unknown	409K	25M	21.22	unknown	TO	failed

TABLE 3
Pipelined ALU with 16 registers: Proving validity.

Hypothesis 1: 'A negative result of SMV means that a property is not valid.'

The circuit from Fig. 16 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. 16 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. 16 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 $\mathbf{1}$ 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. 16 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. 16 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.[12]

F. Additional Experimental Data

To broaden the experimental basis of our evaluation we performed additional experiments both for the pipelined ALU from Fig. 15 with different properties and for a benchmark from the railway transportation domain as described below. All experiments were performed on a Intel Xeon 3.07GHz under Linux with a time limit of 86400 CPU seconds (= 1 day) and a memory limit of 4GB RAM.

Pipelined ALU

In addition to the property considered in Sect. 7.1 we considered the following two properties for the pipelined ALU which were also taken from [5]:

$$AG \bigwedge_{i,j} \Big((EXEX\ \mathbf{R}_{i,j}) \equiv (AXAX\ \mathbf{R}_{i,j})\Big) \tag{20}$$

$$AG \bigwedge_{i,j} \Big((EXEXEX\ \mathbf{R}_{i,j}) \equiv (AXAXAX\ \mathbf{R}_{i,j})\Big) \tag{21}$$

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; $\mathbf{R}_{i,j}$ is the value of the j-th Bit of the i-th register in the register file. Both formulas hold for the complete design. We replaced the ALU (see Fig. 15) by a Black Box, modeled the Black Box outputs by Z_i variables and included them into the state space.

Formula (20): Our results for formula (20) are shown on the left hand side of Tab. 3. The results show that the formula could be proven to be valid (independently from the implementation of the ALU). All pipelined ALUs up to a bit width of 64 could be verified within a few seconds and with a moderate memory consumption. Comparing the relational preimage computation with the functional preimage computation as introduced in Sect. 5.4 we can observe that

12. 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.

functional preimage computation is faster by a factor between 3.9 and 11.7.

Finally, we performed the same experiment using VIS [10] with non-deterministic signals (using 'Lazy Group Sifting' as in our approach), i.e., the ALU was replaced and its outputs were modeled by non-deterministic signals. VIS computes the result that the formula fails on the design, although it is valid (independently from the implementation of the ALU). The reason for this is due to the fact that formula (20) does not fall into the ACTL fragment of CTL. It is known that the abstraction with non-deterministic signals in VIS is only sound for ACTL formulas. When we compare the run times with our tool all the same, we can observe that our tool computes the correct result much faster (both for relational and functional preimage computation). For bit width of 32 and larger VIS runs into the timeout (of 1 CPU day), whereas our tool finishes within seconds.

If we use weaker approximation methods for the Black Box (using Z-variables or Z_i-variables not in the state space) in our approach, then we can neither prove validity of the formula nor disprove realizability.

Formula (21): The results for formula (21) are shown in Tab. 3 on the right hand side. Unfortunately, we can neither prove validity of the formula nor disprove realizability, even if we use our strongest approximation with Z_i-variables in the state space. In contrast to VIS with non-deterministic signals our tool clearly states that the model checking result is unknown, i.e., validity can not be proven and realizability can not be disproven. This information is computed within seconds by our tool, with run times and memory consumption similar to formula (20).

Railway Case Study

We considered a case study based on the rail segment control from the FunkFahrBetrieb (FFB) specification of the Deutsche Bahn, which is closely related to the European Train Control System (ETCS) level 2/3 Movement Authority. The case study models the railway system, consisting of the rail segments and the trains. The access to the rail segments is controlled by the rail segment manager, who grants or denies rail segment requests issued by the trains. The trains send requests for the rail segments before using them (determined by the schedules of the trains). In our benchmark the trains try to reserve up to three rail segments (following in their schedule) in advance; they do not enter a segment prior to obtaining a permission from the rail segment manager. After leaving a segment, a train returns its grant. Here we consider a subset of the German ICE route network with 6 trains and 66 segments. Fig. 17 shows a graphical representation of the segments (represented by nodes) together with their interconnection structure (represented by edges). Each train and each segment is modeled as a finite state machine. The complete system contains 1096 flip flops.

We considered two classes of properties. The first class of properties checks for potential collisions, i.e., it checks whether two trains may be on a segment with the same number. For a pair of trains i and j with $i \neq j$ formula $R1_{i,j}$ is

$$AG\big(\text{seg\_no\_train}_i \neq \text{seg\_no\_train}_j\big). \tag{22}$$

Thus for 6 trains we obtain 15 formulas $R1_{i,j}$. (Three out of these formulas are trivial, since the corresponding trains do not have any common segments on their schedules.)

The second class of properties checks whether in all configurations reachable from the initial state a request of train i for segment j will be granted in some of the successor states. This leads to the formula $R2_{i,j}$:

$$AG\big(\text{request\_train}_i\text{\_seg}_j \rightarrow EF\text{grant\_seg}_j\text{\_train}_i\big). \tag{23}$$

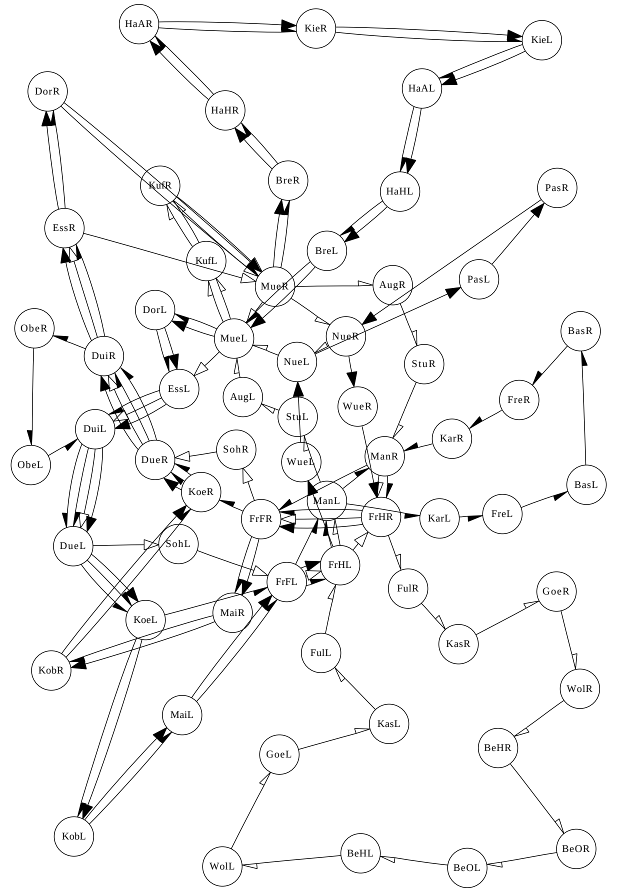


Fig. 17. Railway case study: Track segments are represented by nodes, the routes followed by different trains according to their schedule are represented by edges with different arrow heads for different trains.

Results for formulas $R1_{i,j}$: In the experiments for formulas $R1_{i,j}$ all segments which are not both on the schedule of train i and on the schedule of train j are replaced by Black Boxes (i.e. a segment is replaced by a Black Box, if a collision can never take place on this segment due to the schedules of the trains). In Tab. 4 the name of the formula is given in column 1 (the three trivial formulas mentioned above are omitted), in column 2 the fractions of flip flops remaining in the system after replacements by Black Boxes are given. In a first experiment all Black Box outputs are modeled by Z_i-variables in the state space, in a second experiment they are modeled by Z_i-variables not in the state space. In a third experiment we use a flexible style of modeling: All outputs of Black Boxes corresponding to segments which are neither on the schedule of train i nor on the schedule of train j are modeled by the Z-variable, the remaining outputs by Z_i-variables not in the state space. The results for the first experiment and relational preimage computation are shown in columns 3 and 4 of Tab. 4, the results of the second experiment with relational preimage computation in columns 5 and 6, and the results of the third experiment and relational preimage computation in columns 7 and 8. The corresponding results for functional preimage computation are shown in columns 9–14. Whereas for relational preimage computation there are still a few timeouts in the table, all problems could be solved using functional preimage computation and the validity of the formulas could be proven. In most cases the run times decrease when weaker approximations are used: For relational preimage computation the run times in column 6 are better than those in column 4 in 11 out of 12 cases (for $R1_{3,5}$ and $R1_{5,6}$ even much

Formula	remain. flip flops	Relational preimage computation						Functional preimage computation						MCAIGER		PureSAT
		Z_i in state space		Z_i not in state space		mixed Z/Z_i		Z_i in state space		Z_i not in state space		mixed Z/Z_i				
		mem	time	mem	time	mem	time	mem	time	mem	time	mem	time	result	step	time
$R1_{1,2}$	20.8%	26M	101.1	26M	95.3	11M	36.6	21M	83.8	21M	83.0	6700K	32.7	TO	46	MO
$R1_{1,3}$	20.8%	47M	564.0	44M	132.7	37M	96.1	49M	227.5	48M	193.1	51M	115.9	MO	29	MO
$R1_{1,4}$	18.2%	48M	137.2	27M	120.2	25M	102.8	35M	97.6	37M	100.8	42M	80.8	TO	50	MO
$R1_{1,5}$	18.2%	27M	118.7	27M	100.4	12M	53.5	22M	94.8	22M	97.0	27M	48.1	MO	31	MO
$R1_{1,6}$	18.2%	27M	105.0	27M	102.7	13M	58.0	21M	89.4	21M	86.7	13M	53.1	MO	26	3242
$R1_{2,4}$	18.2%	27M	98.3	26M	93.7	13M	53.1	21M	83.5	21M	82.5	13M	48.2	MO	35	MO
$R1_{3,4}$	51.5%	TO		TO		TO		69M	4387	119M	12783	318M	15969	TO	61	MO
$R1_{3,5}$	31.0%	300M	53755	64M	443.9	59M	416.6	96M	2556	61M	400.6	122M	1958	TO	48	MO
$R1_{3,6}$	28.5%	69M	1511	48M	377.6	52M	165.8	63M	839.3	58M	397.3	68M	737.6	MO	26	MO
$R1_{4,5}$	31.0%	172M	54839	47M	199.3	51M	131.3	91M	2031	98M	856.2	123M	1954	MO	17	MO
$R1_{4,6}$	25.9%	64M	3478	70M	1511	TO		61M	402.2	61M	359.1	57M	278.7	TO	36	MO
$R1_{5,6}$	23.4%	TO		62M	982.0	56M	172.4	48M	351.2	49M	205.7	55M	386.7	TO	47	MO

TABLE 4
Railway case study, Formula $R1_{i,j}$..

Formula	remain. flip flops	Relational preimage		Functional preimage	
		memory	time	memory	time
$R2_{1,0}$	46.6%	65M	1304	60M	405.8
$R2_{2,42}$	23.4%	40M	150.1	24M	88.7
$R2_{3,51}$	54.0%	TO		TO	
$R2_{4,8}$	59.1%	TO		451M	38207
$R2_{5,12}$	33.6%	TO		185M	6170
$R2_{6,2}$	38.7%	TO		104M	1308

TABLE 5
Railway case study, Formula $R2_{i,j}$.

better), the run times in column 8 are better than the run times in column 6 in 10 out of 12 cases. For functional preimage computation the situation is similar (but somewhat less clear than before): the run times in column 12 are better than the run times in column 10 in 9 out of 12 cases, the run times in column 14 are better than those in column 12 in 7 out of 12 cases. The results confirm again that it is possible to verify large non-trivial examples by replacing parts of the design which are not relevant to the considered property by Black Boxes and they confirm the benefit from flexible modeling of Black Box outputs.

Note that we are not able to prove validity, if we choose an even weaker approximation which models all Black Box outputs by the Z-variable.

Since the formulas $R1_{i,j}$ are safety properties, it is possible to check them using SAT solvers as well. We tried both McAiger (version 100810) [62] and PureSAT [17] which is included in VIS. McAiger uses Bounded Model Checking with k–induction [40]. PureSAT also uses k-induction, but for automatically abstracted models; if a counterexample in the abstract model is not concretizable, then abstraction refinement is performed by analyzing the proof of nonexistence of counterexamples of a certain length in the concrete model (see also Sect. 8). Column 15 shows the results of McAiger. All runs exceeded either the time limit (1 CPU day) or the memory limit (4 GB). Column 16 gives the number k of unwindings of the design in McAiger's Bounded Model Checking approach which were analyzed when the time limit / memory limit was exceeded. The results for PureSAT are shown in column 17. PureSAT with its automatic abstraction refinement finishes at least for one of the simpler instances ($R1_{1,6}$), within a CPU time of 54 minutes. In this example the simple strategy used for our tool ("mask out segments which are not both on the schedule of train 1 and on the schedule of train 6") removes the largest number of flip flops among all other instances (896 flip flops out of 1096), leading to a run time of 53.1 CPU seconds for the flexible approach with Z/Z_i modeling and functional preimage computation.

Results for formulas $R2_{i,j}$: In Tab. 5 we show results of formulas $R2_{i,j_i}$ for each train i together with one segment j_i on its schedule. The segment j_i for train i was selected randomly with the additional constraint that it has to be on the schedule of train i. (Otherwise the formula $R2_{i,j_i}$ is trivially true, since the request signal request_train$_i$_seg$_{j_i}$ is constant false, leading to run times of a few seconds in all of these cases.) Here all segments which are not on the schedule of train i were replaced by Black Boxes. The fraction of remaining flip flops in the model is given in column 2 of Tab. 5, it ranges from 23.4% to 59.1% (corresponding to 256 to 648 flip flops). Both run times and memory consumptions for formulas $R2_{i,j}$ are considerably larger than those for formulas $R1_{i,j}$. Nevertheless, the version with functional preimage computation is able to prove validity within the time and memory limits for all but one instance. Validity could be proven using the simplest modeling of Black Box outputs by a single Z-variable.