Symbolic Model Checking for Incomplete Designs

Tobias Nopper
Christoph Scholl

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 201, May 2004

Symbolic Model Checking for Incomplete Designs

Tobias Nopper Christoph Scholl

Institute of Computer Science
Albert-Ludwigs-University Freiburg
D-79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de

We consider the problem of checking whether an incomplete design can still be extended to a complete design satisfying a given CTL formula and whether the property is satisfied for all possible extensions.

Motivated by the fact that well-known model checkers like SMV or VIS produce incorrect results when handling unknowns by using the programs' non-deterministic signals, we present a series of approximate, yet sound algorithms to process incomplete designs with increasing quality and computational resources. Furthermore, we present an exact algorithm to process incomplete designs in which for each unknown area a fixed upper bound on the number of internal states is assumed and an approximate, yet sound method based on this. Finally we give a series of experimental results demonstrating the effectiveness and feasibility of the presented methods.

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

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

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

Some well-known model checking tools like SMV [4] (resp. NuSMV [5]), and VIS [6] provide the definition of non-deterministic signals (see [7, 8, 9]). At first sight, signals coming from unknown areas can be handled as non-deterministic signals, but we will show that modeling by non-deterministic signals is not capable of answering the questions of realizability ('is there a replacement of the Black Boxes so that the overall implementation satisfies a given property?') or validity ('is a given property satisfied for all replacements of the Black Boxes?'). This approach is even not able to provide approximate solutions for realizability or validity. Whereas an *exact* solution to the realizability problem for incomplete designs with several Black Boxes (potentially containing an unrestricted amount of memory) is undecidable in general [10], we will present *approximate* solutions to symbolic model checking for incomplete designs in the first part of our paper. Our algorithms will not give a definite answer in every case, but they are guaranteed to be sound in the sense that they will never give an incorrect answer; they provide proofs of validity and disproofs of realizability. First experimental results given in Sect. 4.3 prove effectiveness and feasibility of these approximate methods.

Our methods are based on symbolic representations of incomplete combinational circuits [11]. Using these representations we provide different methods for approximating the sets of states satisfying a given property φ. During one run of symbolic model checking we compute both underapproximations and overapproximations of the states satisfying φ and we will use them to provide approximate answers for realizability and validity.

The work of Huth et al. [12], which introduced Kripke Modal Transition Systems (KMTSs), comes closest to this approach. Whereas our simplest algorithm can be modeled by using KMTSs, KMTSs are not able to model the fact that the Black Box outputs can not take different values at the same time, a constraint that will be considered in the most advanced algorithm of this section.

Black Boxes in incomplete designs may be seen as Uninterpreted Functions (UIFs) in some sense. UIFs have been used for the verification of pipelined microprocessors [13], where a validity problem is solved under the assumption that both specification and implementation contain the same Uninterpreted Functions. Whereas in [13, 14, 15, 16] a dedicated class of problems for pipelined microprocessors is solved (which is basically reduced to a combinational problem using an inductive argument), we will deal here with arbitrary incomplete sequential circuits and properties given in the full temporal logic CTL.

In the second part of this paper, we additionally present a concept how to perform exact symbolic model checking under the bounded memory assumption, i.e. for each of the Black Boxes a fixed upper bound on the number of internal states is assumed. The algorithm is based on the extraction of the memory out of the Black Boxes and (conceptually) on considering all possible choices for the Black Box instantiations in parallel by means of symbolic methods.

Based on this exact symbolic model checking algorithm, we present another approximate algorithm which is able to prove realizability and to falsify validity by considering simplified Black Boxes. Experimental results given in Sect. 5.5 provide a first evaluation of this method.

The paper is structured as follows: After giving a brief review of symbolic model checking and of representations for incomplete designs in Sect. 2, we will discuss the results of the method handling Black Boxes using non-deterministic signal definitions as provided by SMV and VIS, together with the arising problems in Sect. 3. In Sect. 4, we will introduce several algorithms capable of performing sound and approximate symbolic model checking for incomplete circuits and we will give a series of experimental results demonstrating the effectiveness and feasibility of the presented methods. In Sect. 5, we will introduce an exact algorithm to process incomplete designs in which a fixed upper bound on the number of internal states is assumed for each unknown area and we will present another approximate, yet sound method based on this approach. Again, we evaluate the presented method by a series of experimental results. Sect. 6 concludes the paper.

2 Preliminaries

2.1 Symbolic Model Checking for Complete Designs

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

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

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

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

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

The states of the corresponding Kripke structure are defined as a combination of states and inputs of M. The resulting Kripke structure for M is given by $struct(M) := (S, R, L)$ with:

$$\begin{aligned} S :=& \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \\ R :=& \big\{ \big((\vec{q}, \vec{x}), (\vec{q}\,', \vec{x}\,') \big), \big| \vec{q}, \vec{q}\,' \in \mathbb{B}^{|\vec{q}|}, \vec{x}, \vec{x}\,' \in \mathbb{B}^{|\vec{x}|},\ \delta(\vec{q}, \vec{x}) = \vec{q}\,' \big\} \\ L\big((\vec{q}, \vec{\epsilon})\big) :=& \big\{ x_i \big| \epsilon_i = 1 \big\} \cup \big\{ y_i \big| \lambda_i(\vec{q}, \vec{\epsilon}) = 1 \big\}. \end{aligned}$$

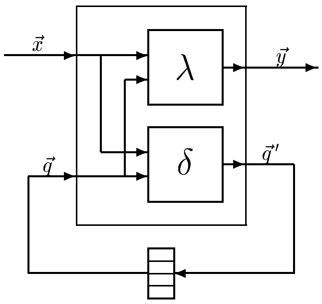


Figure 1: A Mealy automaton.

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

$$
\begin{array}{lcl}
s \models \varphi;\ \varphi \in V & \Longleftrightarrow & \varphi \in L(s) \quad (V = \{x_i | 0 \leq i < |\vec{x}|\} \cup \{y_j | 0 \leq j < |\vec{y}|\}, \\
 & & \qquad\qquad\qquad\qquad \text{the set of atomic propositions}) \\
s \models \neg\varphi & \Longleftrightarrow & s \not\models \varphi \\
s \models (\varphi_1 \vee \varphi_2) & \Longleftrightarrow & s \models \varphi_1 \text{ or } s \models \varphi_2 \\
s \models EX\varphi & \Longleftrightarrow & \exists s' \in S\colon R(s, s') \text{ and } s' \models \varphi \\
s \models EG\varphi & \Longleftrightarrow & s \models \varphi \text{ and } \exists s' \in S\colon R(s, s') \text{ and } s' \models EG\varphi \\
s \models E\varphi_1 U \varphi_2 & \Longleftrightarrow & s \models \varphi_2 \text{ or } \big(\exists s' \in S\colon R(s, s') \text{ and } s \models \varphi_1 \text{ and } s' \models E\varphi_1 U \varphi_2\big)
\end{array}
$$

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

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

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

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

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

a) Fixed point iteration for EG

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

b) Fixed point iteration for EU

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

Figure 2: Fixed point iteration algorithms

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

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

2.2 Incomplete Designs

2.2.1 Representing Incomplete Designs

If parts of a circuit are not yet known or cut off, we have to handle *incomplete designs*. In this section we briefly review symbolic representations of incomplete designs which we will need in Sect. 4.

Unknown parts of the design are combined into so-called 'Black Boxes' (see Fig. 3a for a combinational example with one Black Box).

Consider the combinational part of a sequential circuit (Mealy automaton) defining the transition function δ and the output function λ. For simulating this combinational circuit wrt. some input vector we can make use of the ternary $(0,1,X)$-logic [17, 11]: We assign a value X to each output of the Black Box (since the Black Box outputs are unknown) and we perform a conventional $(0,1,X)$-simulation [18] (see Fig. 3b). If the value of some primary output is X, we do not know the value due to the unknown behavior of the Black Boxes.

For a symbolic representation of the incomplete circuit we model the additional value X by a new variable Z as in [19, 11]. For each output g_i of the incomplete design with primary input variables $x_1, \ldots, x_n$ we obtain a BDD representation of g_i by using a slightly modified version of symbolic simulation with

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

This modified version of symbolic simulation is called *symbolic (0,1,X)-simulation*, see Fig. 3c for an example.

Since $(0,1,X)$-simulation can not distinguish between unknown values at different Black Box outputs, some information is lost in symbolic $(0,1,X)$-simulation. This problem can be solved at the cost of additional variables: Instead of using the same variable Z for all

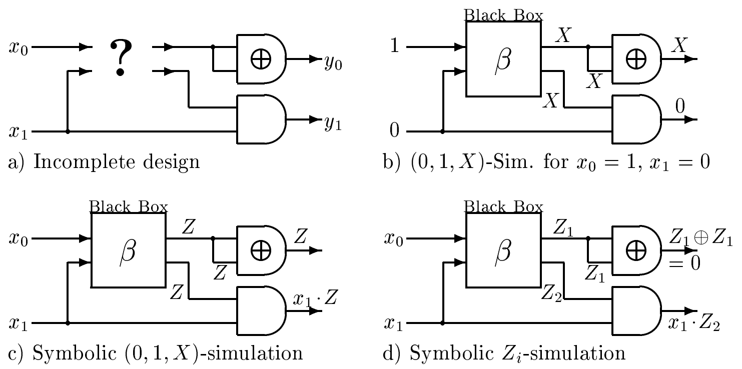


Figure 3: Incomplete design

Black Box outputs, we introduce a new variable Z_i for each Black Box output and perform a (conventional) symbolic simulation. This approach was called symbolic Z_i-simulation in [11]. Fig. 3d shows an example for symbolic Z_i-simulation. (Note that the first output can now be shown to be constant 0.)

In Sect. 4 we will use symbolic $(0, 1, X)$-simulation and symbolic Z_i-simulation to approximate transition functions and output functions of incomplete sequential circuits.

Please note that in contrast to [11], we will consider Black Boxes that can be replaced not only by combinational, but also by sequential circuits, so that for two states in a computation path that generate the same Black Box input, the Black Box may answer with different outputs.

2.2.2 Realizability and Validity

In Sect. 4 and Sect. 5 we will present methods performing (approximate) symbolic model checking for incomplete designs. We will consider two types of questions:

1. Is there a replacement of the Black Boxes in the incomplete design, so that the resulting circuit satisfies a given CTL formula φ? If this is true, then the property φ is called *realizable* for the incomplete design. The corresponding decision problem is called *realizability* problem.

2. Is a CTL formula φ satisfied for all possible replacements of the Black Boxes? If this is the case, then φ is *valid* for the incomplete design; the corresponding decision problem is denoted as *validity* problem.

3 Model Checking for Incomplete Designs using Non-Deterministic Signals

Well-known CTL model checkers such as SMV and VIS provide so-called 'non-deterministic assignments' resp. 'non-deterministic signals' to model non-determinism [7, 8, 9]. At

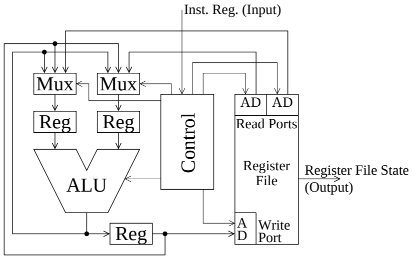


Figure 4: Pipelined ALU

first sight it appears to be advisable using non-deterministic signals for handling Black Box outputs, since the functionality of Black Boxes is not known. In this section we motivate our approach by the observation that non-deterministic signals lead to incorrect results when used for model checking of incomplete designs. We will show that they even can not be used to obtain approximate results by analyzing two small examples.

Before doing so, we will report on a larger and more familiar example showing the same problems. Interestingly, incorrect results of SMV (resp. VIS) due to non-deterministic signals can be observed for the well-known pipelined ALU circuit from [3] (see Fig. 4). In [3], Burch et al. showed by symbolic model checking that (among other CTL formulas) the following formulas are satisfied for the pipelined ALU (the formulas essentially say that the content of the register file $\mathbf{R}$ two (resp. three) clock cycles in the future is uniquely determined by the current state of the system):

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

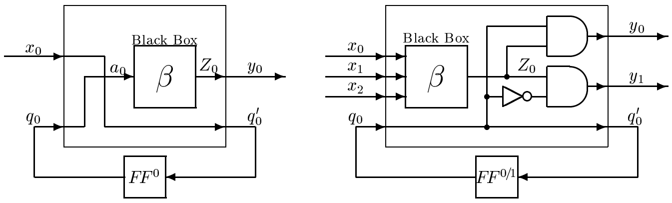
$$AG\big((EX)^3\mathbf{R} \equiv (AX)^3\mathbf{R}\big) \tag{2}$$

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

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

Hypothesis 1: A negative result of SMV means that a property is not valid. Figure 5a shows a counterexample for this hypothesis: If we substitute the Black Box output by a

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



a) First counterexample b) Second counterexample

Figure 5: Counterexamples

non-deterministic signal, SMV provides the result that $\varphi_1 = AG(AXy_0 \vee AX\neg y_0)$ is *not* satisfied. Now consider two finite primary input sequences 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) 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. 5b and the CTL formula $\varphi_2 = EX(EGy_0 \vee AGy_1)$. We assume that the flip flop is initialized by 0. If we replace the Black Box output by a non-deterministic signal, SMV provides the result that φ_2 is *not* satisfied. However, it is easy to see that the formula is satisfied if the Black Box is substituted with the constant **1** function; so the property is realizable. Thus, a negative result of SMV does *not* mean that a property is not realizable.

Hypothesis 3: A positive result of SMV means that a property is valid. Again, we consider the example shown in Fig. 5b and the CTL formula $\varphi_2 = EX(EGy_0 \vee AGy_1)$, yet this time we assume that the flip flop is initialized by 1. If we substitute the Black Box output by a non-deterministic signal, SMV provides the result that φ_2 *is* satisfied. Though, it is easy to see that the formula is not satisfied if the Black Box is substituted with the constant **0** function; so the property 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. 5a in combination with $\varphi_3 = \neg\varphi_1 = \neg AG(AXy_0 \vee AX\neg y_0)$. Again, we assume the Black Box output to be a non-deterministic signal and we verify the circuit using SMV, which provides the result that φ_3 *is* satisfied. However, since property φ_3 is the negation of property φ_1 which has been proven to be valid when considering the first hypothesis, it is quite obvious that φ_3 is not realizable. Thus, a positive result of SMV does *not* mean that a property is realizable.

Conclusion

Using non-deterministic 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.[2]

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

4 Approximate Symbolic Model Checking for Incomplete Designs

4.1 Basic Principle

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

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

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

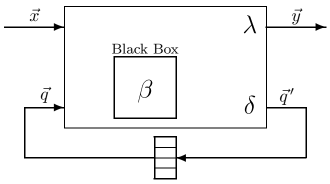


Figure 6: Mealy automaton with Black Box

4.2 Approximations

For reasons of efficiency we will not compute exact sets $Sat_E^{\mathrm{ex}}(\varphi)$ and $Sat_A^{\mathrm{ex}}(\varphi)$. Instead we will compute *approximations* $Sat_E(\varphi)$ and $Sat_A(\varphi)$ of these sets. To be more precise we will compute overapproximations $Sat_E(\varphi) \supseteq Sat_E^{\mathrm{ex}}(\varphi)$ of $Sat_E^{\mathrm{ex}}(\varphi)$ and underapproximations $Sat_A(\varphi) \subseteq Sat_A^{\mathrm{ex}}(\varphi)$ of $Sat_A^{\mathrm{ex}}(\varphi)$.

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

Approximations of $Sat_E(\varphi)$ and $Sat_A(\varphi)$ will be computed based on an approximate transition relation and on approximate output functions for the corresponding Mealy automaton M. In incomplete designs we have Black Boxes in the functional block defining the transition function δ and the output function λ (see Fig. 6). For this reason there are two types of transitions for the automaton: We have

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

We will work with two types of approximations of the transition relation $\chi_R(\vec{q}, \vec{x}, \vec{q}')$: An underapproximation $\chi_{R_A}(\vec{q}, \vec{x}, \vec{q}')$ will only contain fixed transitions and an overapproximation $\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}')$ will contain at least all possible transitions (of course, this includes all fixed transitions).

In the same manner we will approximate the sets of states $Sat(y_i)$ in which the output value y_i of λ_i is true:

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

[2] 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 at the end of the evaluation.

Based on these approximations χ_{R_A}, χ_{R_E}, $Sat_A(y_i)$, and $Sat_E(y_i)$ we will compute the underapproximations $Sat_A(\varphi)$ and overapproximations $Sat_E(\varphi)$ mentioned above for arbitrary CTL formulas φ.

In the following we will present different approximate methods which will (among other things) differ from the accuracy of approximating transition relation and output functions. More exact methods will identify more fixed transitions and less possible transitions. We will make use of symbolic $(0,1,X)$-simulation and symbolic Z_i-simulation for computing δ and λ as described in Section 2.

4.2.1 Symbolic Z-Model Checking

We apply symbolic $(0,1,X)$-simulation (see Section 2) for computing δ and λ. Thus, we introduce a new variable Z, which is assigned to each output of a Black Box and symbolic $(0,1,X)$-simulation provides symbolic representations of functions $\lambda_i(\vec{q},\vec{x},Z)$ and $\delta_j(\vec{q},\vec{x},Z)$.

Output functions: If $\lambda_i|_{\vec{q}=\vec{q}^{\,\mathrm{fix}},\,\vec{x}=\vec{x}^{\,\mathrm{fix}}} = 1$ for some state $(\vec{q}^{\,\mathrm{fix}},\vec{x}^{\,\mathrm{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|}$, then we know that λ_i is 1 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\mathrm{fix}},\vec{x}^{\,\mathrm{fix}})$ into $Sat_A(y_i)$ and $Sat_E(y_i)$. If $\lambda_i|_{\vec{q}=\vec{q}^{\,\mathrm{fix}},\,\vec{x}=\vec{x}^{\,\mathrm{fix}}} = Z$, then the output λ_i may or may not be equal to 1 and thus, we include $(\vec{q}^{\,\mathrm{fix}},\vec{x}^{\,\mathrm{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$. This leads to the following symbolic representations:

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

Transition functions: An analogous argumentation leads to fixed transitions and possible transitions of χ_R, since the outputs of the transition functions may be definitely 1 or 0 (independently from the Black Boxes) or they may be unknown: For χ_{R_A}, representing only fixed transitions we obtain

$$\chi_{R_A}(\vec{q},\vec{x},\vec{q}\,') = \Big(\prod_{i=0}^{|\vec{q}|-1} \forall Z\big(\delta_i(\vec{q},\vec{x},Z) \equiv q_i'\big)\Big)$$

and for χ_{R_E} representing at least all possible transitions we obtain

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

Note that χ_{R_A} defined in this way underapproximates the set of all fixed transitions due to well-known deficiencies of $(0,1,X)$-simulation [11] and χ_{R_E} overapproximates the set of all possible transitions (the same is true for $\chi_{Sat_A(y_i)}$ and $\chi_{Sat_E(y_i)}$, respectively).

In order to compute $Sat_A(\varphi)$ and $Sat_E(\varphi)$ recursively for arbitrary CTL formulas we need rules to evaluate operators EX, $\neg$, $\vee$, EG and EU.

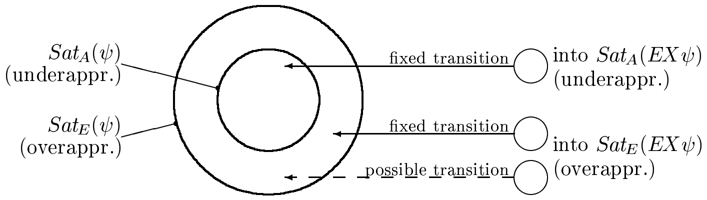


Figure 7: Evaluation of $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$

Computing $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$: Given $Sat_A(\psi)$, the set of states which definitely satisfy ψ for all Black Box replacements, we include into $Sat_A(EX\psi)$ all states with a fixed transition to a state in $Sat_A(\psi)$. It is easy to see that these states definitely satisfy $EX\psi$, independently from the replacement of the Black Boxes. Likewise, we include all the states into $Sat_E(EX\psi)$ which have a possible transition to a state in $Sat_E(\psi)$. Fig. 7 illustrates the sets. Thus, we have

$$\chi_{Sat_A(EX\psi)}(\vec{q},\vec{x}) = \exists \vec{q}' \exists \vec{x}' \big(\chi_{R_A}(\vec{q},\vec{x},\vec{q}') \cdot \big(\chi_{Sat_A(\psi)}|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}')\big)$$

and

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

Computing $Sat_A(\neg\psi)$ and $Sat_E(\neg\psi)$: $Sat_E(\psi)$ is an overapproximation of all states in which ψ *may be* satisfied for some Black Box replacement. Thus, we do know that for an arbitrary state in $\mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \setminus Sat_E(\psi)$ there is no Black Box replacement so that ψ is satisfied in this state or, equivalently, $\neg\psi$ is definitely satisfied in this state for all Black Box replacements. This means that we can use $\mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \setminus Sat_E(\psi)$ as an underapproximation $Sat_A(\neg\psi)$. Since an analogous argument holds for $Sat_A(\psi)$ and $Sat_E(\neg\psi)$ we define

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

Evaluating $\vee$, EG and EU: It is easy to see that $\chi_{Sat_A(\varphi_1\vee\varphi_2)} = \chi_{Sat_A(\varphi_1)} \vee \chi_{Sat_A(\varphi_2)}$ and $\chi_{Sat_E(\varphi_1\vee\varphi_2)} = \chi_{Sat_E(\varphi_1)} \vee \chi_{Sat_E(\varphi_2)}$. Moreover, $\varphi = EG\psi$ and $\varphi = E\psi_1 U \psi_2$ can be evaluated by standard fixed point iterations according to Figures 2a and 2b based on the evaluation of EX defined above (two separate fixed point iterations for Sat_A and Sat_E).

Parallel Evaluation: Altogether we obtain an algorithm to compute approximations for $Sat_A(\varphi)$ and $Sat_E(\varphi)$. According to the arguments given at the beginning of this section we need just $Sat_E(\varphi)$ to falsify realizability and we need just $Sat_A(\varphi)$ to prove validity. However, evaluation of negation shows that it is advisable to compute both $Sat_A(\varphi)$ and $Sat_E(\varphi)$ in parallel, since we need $Sat_A(\psi)$ to compute $Sat_E(\neg\psi)$ and we need $Sat_E(\psi)$ to compute $Sat_A(\neg\psi)$. Note that we do not need to perform two separate model checking runs to compute $Sat_E(\varphi)$ and $Sat_A(\varphi)$. By using an additional encoding variable e and defining $\chi_R = \chi_{R_A} + e \cdot \chi_{R_E}$, we can easily combine the two computations of $\chi_{Sat_A(\varphi)}$ and $\chi_{Sat_E(\varphi)}$ into one computation for $\chi_{Sat(\varphi)} = \overline{e} \cdot \chi_{Sat_A(\varphi)} + e \cdot \chi_{Sat_E(\varphi)}$ ($= \chi_{Sat_A(\varphi)} + e \cdot \chi_{Sat_E(\varphi)}$ due to $\chi_{Sat_A(\varphi)} \leq \chi_{Sat_E(\varphi)}$).

Taken together, with given δ and λ, Symbolic Z-model checking can be performed by using the following set of rules:

$$\begin{aligned}
\chi_{Sat(x_i)}(\vec{q},\vec{x},e) &:= x_i \\
\chi_{Sat(y_i)}(\vec{q},\vec{x},e) &:= \overline{e}\cdot\chi_{Sat_A(y_i)}(\vec{q},\vec{x}) + e\cdot\chi_{Sat_E(y_i)}(\vec{q},\vec{x}) \\
&= \overline{e}\cdot\forall Z\big(\lambda_i(\vec{q},\vec{x},Z)\big) + e\cdot\exists Z\big(\lambda_i(\vec{q},\vec{x},Z)\big) \\
\chi_{Sat(\neg\varphi)}(\vec{q},\vec{x},e) &:= \overline{e}\cdot\chi_{Sat_A(\neg\varphi)}(\vec{q},\vec{x}) + e\cdot\chi_{Sat_E(\neg\varphi)}(\vec{q},\vec{x}) \\
&= \overline{e}\cdot\overline{\chi_{Sat_E(\varphi)}(\vec{q},\vec{x})} + e\cdot\overline{\chi_{Sat_A(\varphi)}(\vec{q},\vec{x})} \\
&= \overline{\big(e\cdot\chi_{Sat_A(\varphi)}(\vec{q},\vec{x}) + \overline{e}\cdot\chi_{Sat_E(\varphi)}(\vec{q},\vec{x})\big)} \\
&= \big(\overline{\chi_{Sat(\varphi)}}|_{e\leftarrow\overline{e}}\big)(\vec{q},\vec{x},e) \\
\chi_{Sat((\varphi_1\vee\varphi_2))}(\vec{q},\vec{x},e) &:= \overline{e}\cdot\chi_{Sat_A(\varphi_1\vee\varphi_2)}(\vec{q},\vec{x}) + e\cdot\chi_{Sat_E((\varphi_1\vee\varphi_2))}(\vec{q},\vec{x}) \\
&= \chi_{Sat(\varphi_1)}(\vec{q},\vec{x},e) + \chi_{Sat(\varphi_2)}(\vec{q},\vec{x},e) \\
\chi_{Sat(EX\varphi)}(\vec{q},\vec{x},e) &:= \exists\vec{q}'\exists\vec{x}'\big(\chi_R(\vec{q},\vec{x},\vec{q}',e)\cdot\big(\chi_{Sat(\psi)}|_{\substack{\vec{q}\leftarrow\vec{q}'\\ \vec{x}\leftarrow\vec{x}'}}\big)(\vec{q}',\vec{x}',e)\big)
\end{aligned}$$

with

$$\begin{aligned}
\chi_R(\vec{q},\vec{x},\vec{q}',e) &:= \overline{e}\cdot\chi_{R_A}(\vec{q},\vec{x},\vec{q}') + e\cdot\chi_{R_E}(\vec{q},\vec{x},\vec{q}') \\
&= \overline{e}\cdot\Big(\prod_{i=0}^{|\vec{q}|-1}\forall Z\big(\delta_i(\vec{q},\vec{x},Z)\equiv q_i'\big)\Big) + e\cdot\Big(\prod_{i=0}^{|\vec{q}|-1}\exists Z\big(\delta_i(\vec{q},\vec{x},Z)\equiv q_i'\big)\Big).
\end{aligned}$$

Again, for EG and EU, the standard fixed point iterations according to Figures 2a and 2b were used. The result of the recursive calculation can then be evaluated as follows:

$$\begin{aligned}
\forall\vec{x}\big(\chi_{Sat(\varphi)}|_{\substack{\vec{q}=\vec{q}^0\\ e=0}}\big) = 1 &\implies \varphi \text{ is valid} \\
\forall\vec{x}\big(\chi_{Sat(\varphi)}|_{\substack{\vec{q}=\vec{q}^0\\ e=1}}\big) = 0 &\implies \varphi \text{ is not realizable}
\end{aligned}$$

Example: Again, we consider the incomplete circuit shown in Fig. 5b. It is quite obvious that in every state at least one of the two primary outputs y_0 and y_1 has to be 0 independently from the Black Box implementation. This is expressed by the CTL formula $\varphi := AG(\neg y_0 \vee \neg y_1)$. By recursively evaluating the subformulas using the approximate algorithm described above, we obtain $\chi_{Sat_A(\varphi)} = \chi_{Sat_E(\varphi)} = 1$ and thus we can prove that the formula is satisfied for all possible replacements of the Black Box.

Symbolic Z-simulation of λ and δ returns:

$$\begin{aligned}
\lambda(\vec{q},\vec{x},Z) &= (q_0\cdot Z, \overline{q}_0\cdot Z) \\
\delta(\vec{q},\vec{x},Z) &= (q_0)
\end{aligned}$$

This leads to the following transition function:

$$\begin{aligned}\chi_R(\vec{q},\vec{x},\vec{q}',e) &= \overline{e}\cdot\Big(\prod_{i=0}^{|\vec{a}|-1}\forall Z\big(\delta_i(\vec{q},\vec{x},Z)\equiv q_i'\big)\Big)+e\cdot\Big(\prod_{i=0}^{|\vec{a}|-1}\exists Z\big(\delta_i(\vec{q},\vec{x},Z)\equiv q_i'\big)\Big)\\ &= (q_0\equiv q_0')\end{aligned}$$

Recursive calculation of $\chi_{Sat(AG(\neg y_0\vee\neg y_1))}$:

$$\begin{aligned}\chi_{Sat(y_0)}(\vec{q},\vec{x},e) &= \overline{e}\cdot\big(\forall Z\lambda_0(\vec{q},\vec{x},Z)\big)+e\cdot\big(\exists Z\lambda_0(\vec{q},\vec{x},Z)\big)=e\cdot q_0\\ \chi_{Sat(\neg y_0)}(\vec{q},\vec{x},e) &= \big(\overline{\chi_{Sat(y_0)}}|_{e\leftarrow\overline{e}}\big)(\vec{q},\vec{x},e)=\overline{q}_0+e\\ \chi_{Sat(y_1)}(\vec{q},\vec{x},e) &= \overline{e}\cdot\big(\forall Z\lambda_1(\vec{q},\vec{x},Z)\big)+e\cdot\big(\exists Z\lambda_1(\vec{q},\vec{x},Z)\big)=e\cdot\overline{q}_0\\ \chi_{Sat(\neg y_1)}(\vec{q},\vec{x},e) &= \big(\overline{\chi_{Sat(y_0)}}|_{e\leftarrow\overline{e}}\big)(\vec{q},\vec{x},e)=q_0+e\\ \chi_{Sat((\neg y_0\vee\neg y_1))}(\vec{q},\vec{x},e) &= \chi_{Sat(\neg y_0)}(\vec{q},\vec{x},e)+\chi_{Sat(\neg y_1)}(\vec{q},\vec{x},e)=\overline{q}_0+e+q_0+e=1\\ \chi_{Sat(AG(\neg y_0\vee\neg y_1))}(\vec{q},\vec{x},e) &= 1\end{aligned}$$

Evaluation:

$$\forall\vec{x}\big(\chi_{Sat(AG(\neg y_0\vee\neg y_1))}|_{\substack{\vec{q}=\vec{q}^0\\ e=0}}\big)=1\quad\Longrightarrow\quad\varphi\text{ is valid}$$

4.2.2 Symbolic Z_i-Model Checking

We obtain a second and more accurate approximation algorithm by replacing symbolic $(0,1,X)$-simulation by symbolic Z_i-simulation. In symbolic Z_i-simulation we introduce a new variable Z_i for each output of a Black Box. The output functions $\lambda_i(\vec{q},\vec{x},\vec{Z})$ and transition functions $\delta_j(\vec{q},\vec{x},\vec{Z})$ will now depend on a vector $\vec{Z}$ of additional variables. As in the previous section, we include a state $(\vec{q}^{\,\mathrm{fix}},\vec{x}^{\,\mathrm{fix}})\in\mathbb{B}^{|\vec{q}|\times|\vec{x}|}$ into $Sat_A(y_i)$ iff $\lambda_i|_{\vec{q}=\vec{q}^{\,\mathrm{fix}},\,\vec{x}=\vec{x}^{\,\mathrm{fix}}}=1$ and we include it into $Sat_E(y_i)$ iff $\lambda_i|_{\vec{q}=\vec{q}^{\,\mathrm{fix}},\,\vec{x}=\vec{x}^{\,\mathrm{fix}}}=1$ or $\lambda_i|_{\vec{q}=\vec{q}^{\,\mathrm{fix}},\,\vec{x}=\vec{x}^{\,\mathrm{fix}}}$ depends on the variables $\vec{Z}$. The transition relation is computed accordingly. The advantage of symbolic Z_i-simulation lies in the fact that the cofactors mentioned above may be 1 or 0 whereas the corresponding cofactors of $(0,1,X)$-simulation are equal to Z. In general this leads to smaller overapproximations $Sat_E(\varphi)$ and larger underapproximations $Sat_A(\varphi)$. The formulas for a recursive evaluation of a CTL formula are similar to the previous section (just replace Z by $\vec{Z}$).

An additional improvement of approximations can be obtained by replacing

$$\chi_R(\vec{q},\vec{x},\vec{q}',e)=\overline{e}\cdot\Big(\prod_{i=0}^{|\vec{q}|-1}\forall\vec{Z}\big(\delta_i(\vec{q},\vec{x},\vec{Z})\equiv q_i'\big)\Big)+e\cdot\Big(\prod_{i=0}^{|\vec{q}|-1}\exists\vec{Z}\big(\delta_i(\vec{q},\vec{x},\vec{Z})\equiv q_i'\big)\Big)$$

by

$$\chi_R(\vec{q},\vec{x},\vec{q}',e)=\overline{e}\cdot\forall\vec{Z}\Big(\prod_{i=0}^{|\vec{q}|-1}\big(\delta_i(\vec{q},\vec{x},\vec{Z})\equiv q_i'\big)\Big)+e\cdot\exists\vec{Z}\Big(\prod_{i=0}^{|\vec{q}|-1}\big(\delta_i(\vec{q},\vec{x},\vec{Z})\equiv q_i'\big)\Big)$$

4.2.3 Symbolic Output Consistent Z_i-Model Checking

In this section we will further improve the accuracy of the approximations presented in the last section. Again, we will use the incomplete circuit in Fig. 5b (with flip flop initialized to 0) to motivate the need for an improvement. Consider the CTL formula $EF(y_1 \wedge \neg y_1)$. It is easy to see that the algorithm given in the last section is neither able to prove validity nor falsify realizability for the given incomplete design and the given formula, since the output y_1 will be 0 or 1 depending on the output of the Black Box. However, it is clear that there will be no time during the computation when y_1 is both true and false. This problem can only be solved if we change our state space by including the Black Box outputs into the states of the Kripke structure, i.e. the state space is extended from $(\vec{q}, \vec{x})$ to $(\vec{q}, \vec{x}, \vec{Z})$. In this way the Black Box output values $\vec{Z}$ are constant within each single state and therefore in our example y_1 will have a fixed value for each state.

Considering a state $(\vec{q}, \vec{x})$ of the original state space, the state necessarily satisfies φ, if $\chi_{Sat_A(\varphi)}(\vec{q}, \vec{x}, \vec{Z})$ is true for every possible value of $\vec{Z}$ and the state does not possibly satisfy φ, if $\chi_{Sat_E(\varphi)}(\vec{q}, \vec{x}, \vec{Z})$ is false for any possible value of $\vec{Z}$.

We will now describe how to perform symbolic model checking on the extended state space; we will call this method symbolic output consistent Z_i-model checking. The evaluation of $\chi_{Sat(y_i)}$ is straight-forward for a extended state $(\vec{q}, \vec{x}, \vec{Z})$, since the Black Box output is part of the state now and thus it is possible to directly evaluate whether the output y_i is true or false in this state:

$$\chi_{Sat(y_i)}(\vec{q}, \vec{x}, \vec{Z}, e) := \lambda_i(\vec{q}, \vec{x}, \vec{Z})$$

Likewise, it is no longer necessary to separate the transition function into a 'fixed' and 'possible' part; there is a transition from $(\vec{q}, \vec{x}, \vec{Z})$ to all $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ with $\delta_i(\vec{q}, \vec{x}, \vec{Z}) = \vec{q}\,'$:

$$\chi_R(\vec{q}, \vec{x}, \vec{Z}, \vec{q}\,') := \prod_{i=0}^{|\vec{q}|-1} \Big(\delta_i(\vec{q}, \vec{x}, \vec{Z}) \equiv q_i'\Big)$$

Yet, we have to adjust the calculation of $\chi_{Sat(EX\psi)}$ for the new state space. For $\chi_{Sat_A(EX\psi)}$, we include all states $(\vec{q}, \vec{x}, \vec{Z})$, for which there exist $\vec{q}\,'$ and $\vec{x}\,'$, so that for all Black Box output values $\vec{Z}\,'$: $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ is a successor of $(\vec{q}, \vec{x}, \vec{Z})$ and $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ satisfies $\chi_{Sat_A(\psi)}$.

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

For $\chi_{Sat_E(EX\psi)}$, we include all states $(\vec{q}, \vec{x}, \vec{Z})$, for which there exist $\vec{q}\,'$ and $\vec{x}\,'$, so that for at least one Black Box output value $\vec{Z}\,'$: $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ is a successor of $(\vec{q}, \vec{x}, \vec{Z})$ and $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ satisfies $\chi_{Sat_E(\psi)}$.

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

Joined together (with $\chi_{Sat(\psi)} = \overline{e} \cdot \chi_{Sat_A(\psi)} + e \cdot \chi_{Sat_E(\psi)}$) we receive:

$$\begin{aligned} \chi_{Sat(EX\psi)}(\vec{q}, \vec{x}, \vec{Z}, e) = {} & \overline{e} \cdot \exists \vec{q}\,' \exists \vec{x}\,' \Big(\chi_R(\vec{q}, \vec{x}, \vec{Z}, \vec{q}\,') \cdot \forall \vec{Z}\,' \big(\chi_{Sat(\psi)}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z} \leftarrow \vec{Z}\,'}}\big)(\vec{q}\,', \vec{x}\,', \vec{Z}\,', e)\Big) + \\ & e \cdot \exists \vec{q}\,' \exists \vec{x}\,' \Big(\chi_R(\vec{q}, \vec{x}, \vec{Z}, \vec{q}\,') \cdot \exists \vec{Z}\,' \big(\chi_{Sat(\psi)}|_{\substack{\vec{q} \leftarrow \vec{q}\,' \\ \vec{x} \leftarrow \vec{x}\,' \\ \vec{Z} \leftarrow \vec{Z}\,'}}\big)(\vec{q}\,', \vec{x}\,', \vec{Z}\,', e)\Big) \end{aligned}$$

The calculation of all remaining CTL operands can be adopted from symbolic Z- resp. Z_i-model checking. The result of the recursive calculation can be evaluated as follows:

$$\begin{aligned}\forall\vec{x}\forall\vec{Z}\big(\chi_{Sat(\varphi)}|_{\substack{\vec{q}=\vec{q}^{\,0}\\ e=0}}\big)=1 &\implies \varphi \text{ is valid}\\ \forall\vec{x}\exists\vec{Z}\big(\chi_{Sat(\varphi)}|_{\substack{\vec{q}=\vec{q}^{\,0}\\ e=1}}\big)=0 &\implies \varphi \text{ is not realizable}\end{aligned}$$

Example: Again, we consider the incomplete circuit shown in Fig. 5b. As mentioned above, the CTL formula $\varphi = EF(y_1 \wedge \neg y_1)$ will not be satisfied for any Black Box replacement and is thus not realizable. Other than symbolic Z- and Z_i-model checking, output consistent Z_i-model checking is able to disprove the realizability of the formula.

Symbolic Z_i-simulation of λ and δ returns:

$$\begin{aligned}\lambda(\vec{q},\vec{x},\vec{Z}) &= (q_0\cdot Z_0, \overline{q}_0\cdot Z_0)\\ \delta(\vec{q},\vec{x},\vec{Z}) &= (q_0)\end{aligned}$$

This leads to the following transition function:

$$\begin{aligned}\chi_R(\vec{q},\vec{x},\vec{q}',\vec{Z}) &= \prod_{i=0}^{|\vec{q}|-1}\Big(\delta_i(\vec{q},\vec{x},\vec{Z})\equiv q_i'\Big)\\ &= (q_0 \equiv q_0')\end{aligned}$$

Recursive calculation of $\chi_{Sat(EF(y_1\wedge\neg y_1))}$:

$$\begin{aligned}\chi_{Sat(y_1)}(\vec{q},\vec{x},\vec{Z},e) &= \lambda_1(\vec{q},\vec{x},\vec{Z}) = \overline{q}_0\cdot Z_0\\ \chi_{Sat(\neg y_1)}(\vec{q},\vec{x},\vec{Z},e) &= \big(\overline{\chi_{Sat(y_1)}}|_{e\leftarrow\overline{e}}\big)(\vec{q},\vec{x},\vec{Z},e) = q_0+\overline{Z}_0\\ \chi_{Sat((y_1\wedge\neg y_1))}(\vec{q},\vec{x},\vec{Z},e) &= \chi_{Sat(y_1)}(\vec{q},\vec{x},\vec{Z},e)\cdot\chi_{Sat(\neg y_1)}(\vec{q},\vec{x},\vec{Z},e) = \overline{q}_0\cdot Z_0\cdot(q_0+\overline{Z}_0)=0\\ \chi_{Sat(EF(y_1\wedge\neg y_1))}(\vec{q},\vec{x},\vec{Z},e) &= 0\end{aligned}$$

Evaluation:

$$\forall\vec{x}\exists\vec{Z}\big(\chi_{Sat(EF(y_1\wedge\neg y_1))}|_{\substack{\vec{q}=\vec{q}^{\,0}\\ e=1}}\big)=0 \implies \varphi \text{ is not realizable}$$

4.3 Experimental Results

To demonstrate the feasibility and effectiveness of the presented methods we implemented a prototype model checker called MIND (Model Checker for Incomplete Designs) based on the BDD package CUDD 2.3.1 [20]. MIND uses 'Lazy Group Sifting' [21], a reordering technique particularly suited for model checking, and partitioned transition functions [22].

For a given incomplete circuit and a CTL formula, MIND first tries to gain information by using symbolic Z-model checking. In the case that no result can yet be obtained, MIND moves on to symbolic Z_i-model checking and later − if necessary − to symbolic output consistent Z_i-model checking.

<table>
<tr><th></th><th colspan="5">No Black Boxes</th><th colspan="5">Adder and multiplier replaced by Black Boxes</th><th colspan="5">Adder, multiplier and 12 registers replaced by Black Boxes</th></tr>
<tr><th>word width</th><th>BDD vars</th><th>memory used</th><th>BDD nodes</th><th>RO time</th><th>time</th><th>BDD vars</th><th>memory used</th><th>BDD nodes</th><th>RO time</th><th>time</th><th>BDD vars</th><th>memory used</th><th>BDD nodes</th><th>RO time</th><th>time</th></tr>
<tr><td>2</td><td>115</td><td>15648180</td><td>144681</td><td>16.54</td><td>18.95</td><td>117</td><td>8875028</td><td>88166</td><td>7.29</td><td>8.84</td><td>69</td><td>7691172</td><td>48443</td><td>1.68</td><td>2.44</td></tr>
<tr><td>4</td><td>191</td><td>47698020</td><td>268028</td><td>128.12</td><td>199.09</td><td>193</td><td>43750260</td><td>429830</td><td>215.66</td><td>274.99</td><td>97</td><td>14688292</td><td>105926</td><td>22.92</td><td>30.45</td></tr>
<tr><td>6</td><td>267</td><td>52345860</td><td>1307470</td><td>824.36</td><td>973.38</td><td>269</td><td>14856116</td><td>110846</td><td>27.92</td><td>30.89</td><td>125</td><td>14946340</td><td>66176</td><td>12.10</td><td>20.65</td></tr>
<tr><td>8</td><td>343</td><td>63229572</td><td>2233159</td><td>1804.83</td><td>2286.06</td><td>345</td><td>28912788</td><td>118064</td><td>54.92</td><td>61.97</td><td>153</td><td>39908148</td><td>107613</td><td>18.24</td><td>48.87</td></tr>
<tr><td>10</td><td colspan="5" rowspan="6">more than 12.000 sec.</td><td>421</td><td>47499956</td><td>305772</td><td>243.77</td><td>319.97</td><td>181</td><td>37586452</td><td>139006</td><td>26.38</td><td>68.68</td></tr>
<tr><td>12</td><td>497</td><td>41327748</td><td>120268</td><td>56.37</td><td>74.45</td><td>209</td><td>49600356</td><td>74954</td><td>14.92</td><td>80.60</td></tr>
<tr><td>16</td><td>649</td><td>47498820</td><td>169453</td><td>96.69</td><td>161.03</td><td>265</td><td>48107812</td><td>104683</td><td>30.42</td><td>110.65</td></tr>
<tr><td>32</td><td>1257</td><td>50710356</td><td>220204</td><td>563.31</td><td>763.83</td><td>489</td><td>51717572</td><td>171161</td><td>95.52</td><td>523.28</td></tr>
<tr><td>48</td><td>1865</td><td>65334916</td><td>242826</td><td>692.58</td><td>3603.20</td><td>713</td><td>64384116</td><td>169745</td><td>159.66</td><td>2132.44</td></tr>
<tr><td>64</td><td colspan="5">more than 12.000 sec.</td><td>937</td><td>102258804</td><td>296314</td><td>308.78</td><td>4148.28</td></tr>
</table>

Table 1: Faulty pipelined ALU with 16 registers: Falsifying the realizability of $\varphi_1 = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ using symbolic Z-model checking

For our experiments we used a class of simple synchronous pipelined ALUs similar to the ones presented in [3]. In contrast to [3], our pipelined ALU contains a combinational multiplier (see Figure 8). Since combinational multipliers show exponential size regarding to their width if represented by BDDs [2], symbolic model checking for the complete design can only be performed up to a moderate bit width of the ALU.

In the following we compare a series of complete pipelined ALUs with 16 registers in the register file and varying word width to two incomplete pendants: For the first, the adder and the multiplier are substituted by Black Boxes and for the second, 12 of the 16 registers in the register file are masked out as well.

All experiments were performed on an Intel Pentium4 2.6GHz with 1GB RAM and with a limited runtime of 12.000 seconds.

In a first experiment we inserted an error to the implementation of the XOR operation[3], so it produced incorrect results. We then checked the CTL formula $\varphi_1 = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ which corresponds to formula (1) in [3]. It says that whenever the instruction $\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1$ is given at the inputs, the values in $\mathbf{R}_2$ three clock cycles in the future will be identical to the exclusive-or of $\mathbf{R}_0$ and $\mathbf{R}_1$ in the state two clock cycles in the future ($\mathbf{R}_0$, $\mathbf{R}_1$ and $\mathbf{R}_2$ are the respective first, second and third register in the register file). This property is false for our complete, but *faulty* design, independently of how the adder and multiplier function are implemented. Due to that, φ_1 is not satisfied for any possible Black Box replacement in the incomplete pipelined ALUs, thus not realizable. Note that the Black Boxes lie *inside* the cone of influence for this CTL formula.

In Tab. 1 we give the results for both complete and incomplete pipelined ALUs with varying word width tested with φ_1. For each word width and each pipelined ALU, the table shows the number of BDD variables ('BDD vars'), the peak memory usage, the peak number of BDD nodes, the time spent while reordering the BDD variables ('RO time') and the overall time in CPU seconds.

As mentioned above, a multiplier has a large impact on BDD size and thus on computation time. On account of this, the model checking procedure for complete pipelined ALUs with multipliers of word width beyond 8 bit exceeds the time limit. In contrast to that, the incomplete pipelined ALUs without adder and multiplier can still be verified (using symbolic Z-model checking) and φ_1 can be proven to be unrealizable up to a word width of 48 bit.

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

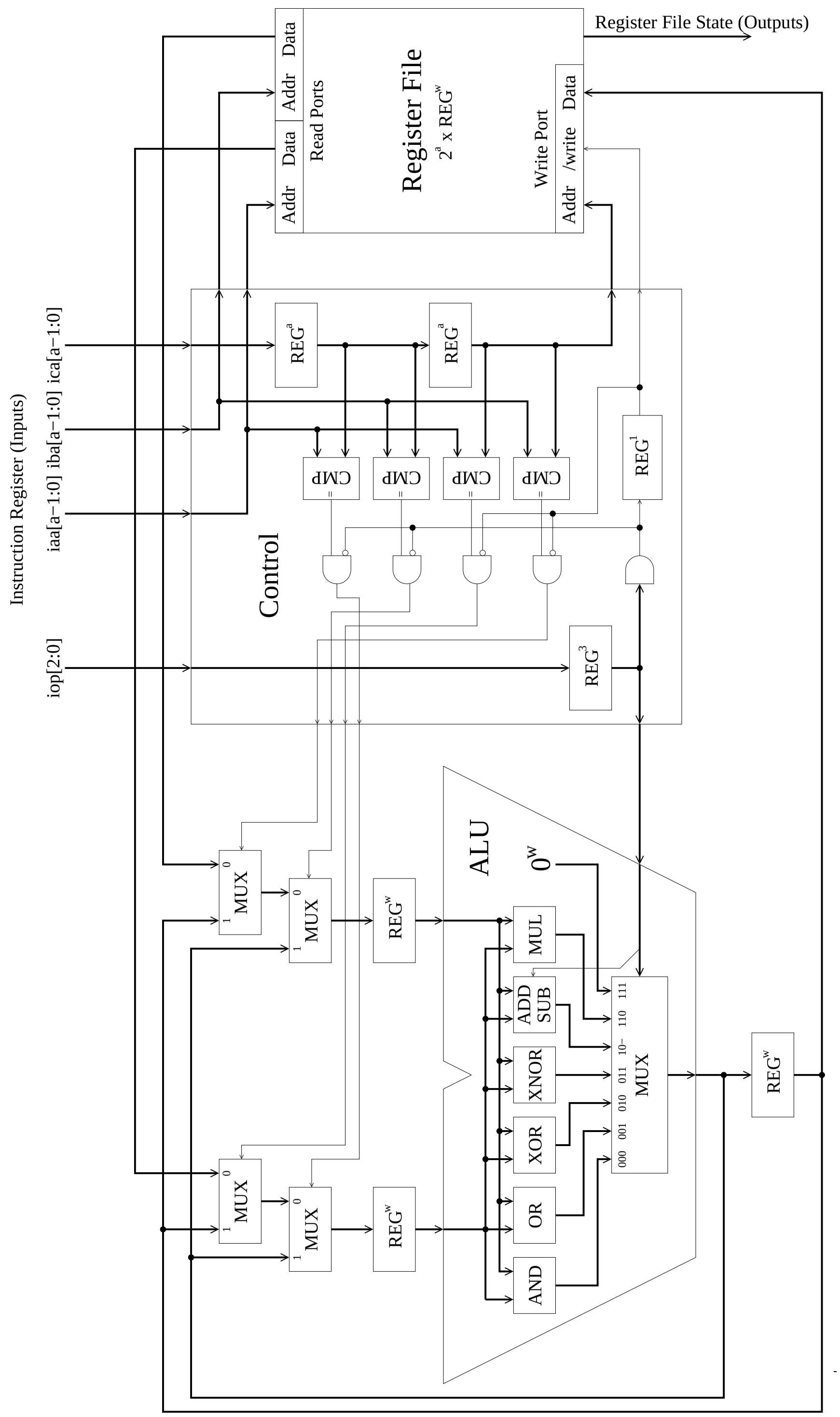


Figure 8: Detailed view of the pipelined ALU used in the experiments.

	No Black Boxes					**Adder and multiplier replaced by Black Boxes**					**Adder, multiplier and 12 registers replaced by Black Boxes**				
word width	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time
2	115	14293796	167732	24.80	27.07	120	18539828	78710	29.68	35.45	96	6289428	40067	3.37	3.99
4	191	44312916	260432	136.43	209.52	200	32040036	280644	141.86	164.02	152	16982804	107882	32.54	42.98
6	267	44974932	677287	354.33	481.75	280	49688116	101814	54.79	115.05	208	48858612	99825	43.09	104.53
8	343	70257572	2062505	1734.78	3149.34	360	48240820	266458	250.10	358.87	264	43916036	184957	87.71	210.44
10	more than 12.000 sec.					440	50132196	334578	381.38	534.24	320	35504804	125120	90.69	162.18
12						520	55120852	311883	1532.51	10279.64	376	45102100	110498	71.10	212.38
16						680	56091444	403086	770.80	1529.96	488	46152532	138276	88.95	297.70
32						1320	69133684	641417	2939.57	7539.18	936	51112436	174608	264.92	1224.07
48						1960	79099060	234256	1462.76	9458.99	1384	54541588	212548	422.95	3270.27
64						more than 12.000 sec.					1832	50217764	299242	827.25	2599.73

Table 2: Correct pipelined ALU with 16 registers: Proving the validity of $\varphi_1 = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ using symbolic output consistent Z_i-model checking

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, making it possible to prove the unrealizability of φ_1 up to a word width of 64 bit. This is mainly due to the decrease of needed BDD variables, caused by the reduction of many q_i and q_i' variables to a single Z variable and the simplification of the transition function, which does no longer depend on the inputs 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.

In a second experiment we considered the same CTL formula as above, yet this time we used a *correct* implementation of the XOR operation. In this case, φ_1 is satisfied for the complete and valid for the incomplete pipelined ALUs.

In Tab. 2 we give the results for both complete and incomplete pipelined ALUs tested with φ_1. In this example, symbolic Z-model checking and symbolic Z_i-model checking were not able to prove the validity of φ_1. However, in all cases the formula could be proved by output consistent Z_i-model checking, which extends the state variables by the Z_i variables. So the values given in Tab. 2 are the overall values for Z-model checking, Z_i-model checking and output consistent Z_i-model checking, since the implementation considers the methods one after the other until one is able to provide a definite result.

The number of BDD variables needed for the incomplete pipelined ALU has increased in comparison to symbolic Z-model checking; this is due to the use of separate Z_i variables for each Black Box output instead of one single Z variable. This can be particularly seen for the pipelined ALU with partially masked register file. But still, the output consistent Z_i-model checking of the incomplete pipelined ALUs outperforms the conventional model checking of the complete version – for the same reasons as given above.

We also checked $\varphi_2 = AG\big((EX)^2\mathbf{R} \equiv (AX)^2\mathbf{R}\big)$ from [3], which is true for the complete design and valid for the incomplete designs, as already mentioned in Sect. 3. This can be proven by using output consistent Z_i-model checking, Tab. 3 shows the results for both complete and incomplete pipelined ALUs tested with φ_2.

	No Black Boxes					Adder and multiplier replaced by Black Boxes					Adder, multiplier and 12 registers replaced by Black Boxes				
word width	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time
2	115	6447172	86134	10.02	10.93	120	5801668	42092	9.89	12.43	96	6589076	17850	1.32	1.82
4	191	27642612	72300	11.29	22.83	200	34270980	145272	56.05	103.96	152	8765844	49606	8.07	15.76
6	267	29000724	33508	6.51	22.09	280	44303300	34719	13.61	68.13	208	26273684	23271	3.95	20.74
8	343	36424244	178326	81.55	273.89	360	46581316	224278	163.71	316.17	264	44918692	135295	35.34	326.56
10	419	15198916	170788	78.06	95.88	440	49120884	250366	150.69	419.51	320	28175428	63962	27.46	58.06
12	495	43453252	424736	181.85	277.18	520	52370660	143689	106.83	2426.27	376	30352164	29256	16.69	51.04
16	more than 12.000 sec.					680	51923348	316126	492.21	2453.67	488	37403732	74058	54.47	130.83
32						1320	55675412	549150	2093.27	6097.25	936	49950532	157746	218.45	608.49
48						1960	55672692	84856	600.54	1734.20	1384	48234196	133424	360.97	1047.16
64						2600	69985524	509326	1780.71	5158.97	1832	46709604	220710	670.19	1874.36

Table 3: Correct pipelined ALU with 16 registers: Proving the validity of $\varphi_2 = AG\big((EX)^2\mathbf{R} \equiv (AX)^2\mathbf{R}\big)$ using symbolic output consistent Z_i-model checking

In a similar manner as for φ_1, the results for φ_2 clearly show that our method outperforms the conventional model checking of the complete version – for the same reasons as given above.

Taken together, the results show that by masking out expensive parts of the pipelined ALU we are still able to provide correct (i.e. sound) and useful results, yet at shorter time and with fewer memory consumption.

5 Symbolic Model Checking for Black Boxes with Bounded Memory

In the last section, we introduced three methods to approximate both $Sat_E^{\text{ex}}(\varphi)$, the set of states, for which there is at least one Black Box replacement so that φ is satisfied, and $Sat_A^{\text{ex}}(\varphi)$, the set of states, for which φ is satisfied for all Black Box replacements. Based on these sets, we were able to provide sound results for falsifying realizability and for proving validity of incomplete designs. Yet, there are formulas, for which it is not possible to provide a result. To give an example, it is not possible to make any statement for the pipelined ALU, in which the adder has been removed, checking formula (2) (see page 7) with the methods presented in the last section.

Due to this, we will consider a different approach in this section: We consider Black Boxes with bounded memory, which means that there is a fixed upper bound on the number of flipflops the possible substitutions are allowed to have. Due to this bounded memory assumption, the number of different Black Box behaviors is finite and thus, it is conceptually possible to calculate $Sat^R(\varphi)$ for each possible replacement R of each Black Box. A CTL formula φ is realizable iff there is a replacement R with all start states lying in $Sat^R(\varphi)$ and a CTL formula φ is valid iff all start states lie in $Sat^R(\varphi)$ for all possible replacements R.

Since the explicit approach is obviously not applicable in practice due to the enormous number of possible Black Box substitutions, we will use symbolic methods to implicitly consider all possible choices for the Black Box substitutions in parallel.

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

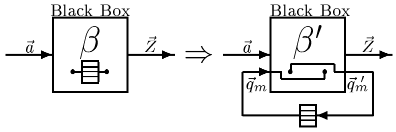


Figure 9: Extracting flipflops from a Black Box with Bounded Memory

we will then consider an approximate version that is able to provide sound results, yet at lower costs. Finally, we will show how to extend the methods to circuits containing multiple Black Boxes with bounded memory.

5.1 Extracting flipflops from a Black Box with Bounded Memory

We consider a Black Box with bounded memory, which means that there is a fixed upper bound on the number of flipflops the possible substitutions are allowed to have; let m be this upper bound. Further we assume that these flipflops are driven by the same clock as the flipflops already included in the circuit.

Given these assumptions, we can separate the flipflops from the Black Box by adding m additional outputs $\vec{q}_m'$ leading to the flipflop inputs and m additional inputs $\vec{q}_m$ going back to the Black Box (Fig. 9); the resulting transformed Black Box is combinational, i.e. the possible substitutions are limited to combinational circuits.

Obviously, for each sequential circuit with a maximum of m flipflops replacing the original Black Box with bounded memory there is a combinational replacement for the newly formed combinational Black Box with m external flipflops and vice versa.

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

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

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

Given an incomplete circuit containing exactly one combinational Black Box, we can divide the combinational part of the Mealy automaton into four parts (see upper part of Fig. 10):

Since the Black Box considered in this section is limited to have only combinational substitutions, we can assume the Black Box to calculate an unknown boolean function $\beta\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$.

Furthermore, let $\alpha\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \to \mathbb{B}^{|\vec{a}|}$ be the boolean function of the circuit part calculating the Black Box inputs $\vec{a}$ and $\lambda\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \to \mathbb{B}^{|\vec{y}|}$ resp. $\delta\colon \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \to \mathbb{B}^{|\vec{q}'|}$ be the boolean functions of the circuit parts calculating the primary output resp. the next state. While α just depends on the primary input $\vec{x}$ and the actual state $\vec{q}$, δ and λ addi-

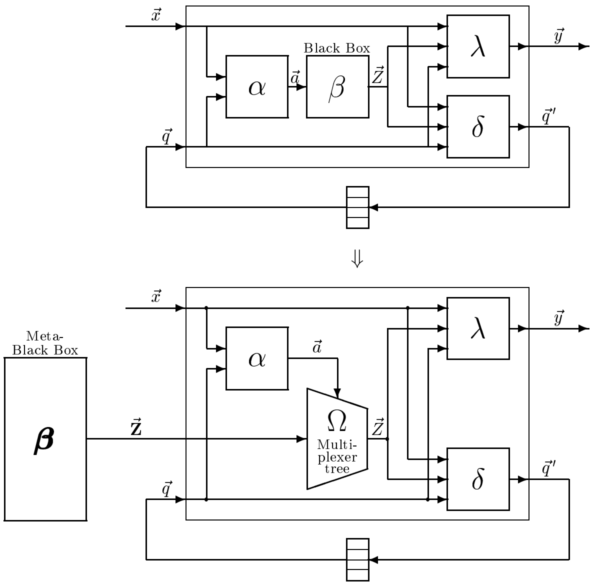


Figure 10: Incomplete circuit with one combinational Black Box and the altered circuit in which the Black Box has been replaced by a Meta Black Box and a readout function.

tionally depend on the Black Box outputs $\vec{Z}$. All these functions can be calculated using symbolic simulation.

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

Every function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ can be described by its corresponding truth table with $m{\cdot}2^n$ entries; likewise, we can describe the Black Box function $\beta\colon \mathbb{B}^{|\vec{a}|} \to |\vec{Z}|$ by a truth table with $|\vec{Z}|{\cdot}2^{|\vec{a}|}$ entries. Since the functionality of the Black Box is constant but unknown, each truth table entry is constant (due to the combinational nature of the Black Box) but unknown as well. So we consider each entry to be a boolean variable $\mathbf{Z}_{i,j} \in \mathbb{B}$ ($0 \le i < 2^{|\vec{a}|}$, $0 \le j < |\vec{Z}|$) with constant, but unknown value. We use $\vec{\mathbf{Z}} := (Z_{0,0}, \ldots, Z_{0,|\vec{Z}|-1}, \ldots, Z_{2^{|\vec{a}|}-1,|\vec{Z}|-1})$ for the whole truth table.

A combinational Black Box with function $\boldsymbol{\beta}\colon \mathbb{B}^0 \to \mathbb{B}^{|\vec{Z}|\cdot 2^{|\vec{a}|}}$ (called 'Meta Black Box' in the following) provides such $|\vec{Z}| \cdot 2^{|\vec{a}|}$ unknown values, which are supposed to be constant, since the Meta Black Box does not have any inputs. Additionally, we define a readout function $\Omega\colon \mathbb{B}^{|\vec{a}|} \times \mathbb{B}^{(|\vec{Z}|\cdot 2^{|\vec{a}|})} \to \mathbb{B}^{|\vec{Z}|}$ that 'reads' the entries in the Meta Black Box depending on the value of $\vec{a}$. Formally, $\Omega_i(\vec{a}, \vec{\mathbf{Z}}) := \mathbf{Z}_{a,i}$, whereas a is the integer value described by the binary number $a_{|\vec{a}|-1} \ldots a_1 a_0$. This readout function can be intuitively realized by using a multiplexer tree.

For each $f\colon \mathbb{B}^{|\vec{a}|} \to \mathbb{B}^{|\vec{Z}|}$ there is exactly one according truth table and vice versa, thus there is exactly one $\mathbf{f}\colon \mathbb{B}^0 \to \mathbb{B}^{|\vec{Z}|\cdot 2^{|\vec{a}|}}$ with $f(\vec{a}) = \Omega(\vec{a}, \mathbf{f})$ for all $\vec{a}$. So we can replace the Black Box β by its corresponding Meta Black Box $\boldsymbol{\beta}$ and the readout function Ω (see lower part of Fig. 10). Obviously, there is a substitution of β so that a property is satisfied iff there is a substitution of $\boldsymbol{\beta}$ as well. Likewise, a property is satisfied for all substitutions of β iff it is satisfied for all substitutions of $\boldsymbol{\beta}$.

Let $\mathbf{M} = (\mathbb{B}^{|\vec{q}|}, \mathbb{B}^{|\vec{x}|}, \mathbb{B}^{|\vec{y}|}, \boldsymbol{\delta}, \boldsymbol{\lambda}, \vec{q}^{\,0})$ be the incomplete Mealy automaton developed from M by the rules given above. $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$ can be computed from λ, δ, α and Ω as follows:

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

Since the combinational Meta Black Box $\boldsymbol{\beta}$ does not have any inputs, its outputs are to carry constant values. Thus the Black Box output values do not change throughout an entire computation run and have a fixed value for each run of the system starting at a certain initial state (which includes the output values $\vec{\mathbf{Z}}$ of the Meta Black Box).

For our exact symbolic model checking, we extend the state space by $\vec{\mathbf{Z}}$. We adjust the recursive calculation of $\chi_{Sat(\varphi)}$ from the original model checking algorithm to match the new state space:

$$\begin{array}{lll}
 & \chi_{Sat(x_i)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = x_i \\
 & \chi_{Sat(y_i)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \boldsymbol{\lambda}_i(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
 & \chi_{Sat(\neg\varphi)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \overline{\chi_{Sat(\varphi)}}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
 & \chi_{Sat((\varphi_1\vee\varphi_2))}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \chi_{Sat(\varphi_1)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) + \chi_{Sat(\varphi_2)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
 & \chi_{Sat(EX\varphi)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \chi_{EX}(\chi_{Sat(\varphi)})(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
 & \chi_{Sat(EG\varphi)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \chi_{EG}(\chi_{Sat(\varphi)})(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
 & \chi_{Sat(E\varphi_1 U\varphi_2)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \chi_{EU}(\chi_{Sat(\varphi_1)}, \chi_{Sat(\varphi_2)})(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \\
\text{with} & \chi_{EX}(\chi_X)(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) & = \exists\vec{q}\,'\exists\vec{x}\,'\Big(\chi_R(\vec{q}, \vec{x}, \vec{\mathbf{Z}}, \vec{q}\,') \cdot \big(\chi_X|_{\substack{\vec{q}\leftarrow\vec{q}\,' \\ \vec{x}\leftarrow\vec{x}\,'}}\big)(\vec{q}\,', \vec{x}\,', \vec{\mathbf{Z}})\Big) \\
\text{and} & \chi_R(\vec{q}, \vec{x}, \vec{\mathbf{Z}}, \vec{q}\,') & = \displaystyle\prod_{i=0}^{|\vec{q}|-1} \big(\boldsymbol{\delta}_i(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) \equiv q_i'\big)
\end{array}$$

Both χ_{EG} and χ_{EU} can be calculated using the original fixed point algorithms shown in Fig. 2.

Since the outputs of the Meta Black Box $\vec{\mathbf{Z}}$ represent the complete truth table of the original Black Box β, thus its whole functionality, we can reduce universal/existential abstraction of the unknown function β to an universal/existential abstraction of $\vec{\mathbf{Z}}$:

$$\begin{array}{lll}
\forall\vec{\mathbf{Z}}\forall\vec{x}\big(\chi_{Sat(\varphi)}|_{\vec{q}=\vec{q}^{\,0}}\big) = 1 & \iff & \varphi \text{ is valid} \\
\exists\vec{\mathbf{Z}}\forall\vec{x}\big(\chi_{Sat(\varphi)}|_{\vec{q}=\vec{q}^{\,0}}\big) = 1 & \iff & \varphi \text{ is realizable}
\end{array}$$

Example We check the circuit shown in Fig. 11a with the CTL formula $\varphi = AFy_0$. First, we replace the combinational Black Box β by its corresponding Meta Black Box $\boldsymbol{\beta}$ representing a

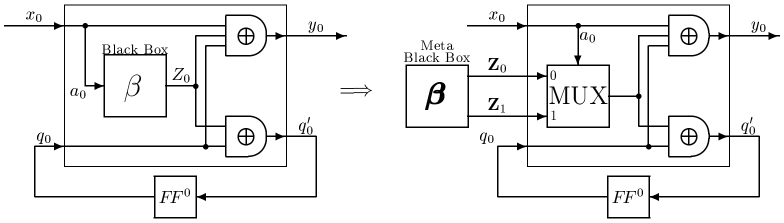


Figure 11: Example for exact symbolic model checking of incomplete designs with one combinational Black Box.

function table with $2^1 = 2$ entries and the readout function Ω – in this case, just a multiplexer. The altered circuit is shown in Fig. 11b. We can now calculate $\boldsymbol{\lambda}$ and $\boldsymbol{\delta}$:

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

This leads to the following transition relation:

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

Calculation of $\chi_{Sat(AFy_0)}$:

$$\chi_{Sat(y_0)}(\vec{q}, \vec{x}, \vec{\mathbf{Z}}) = \big(x_0 \oplus q_0 \oplus (\overline{x}_0 \cdot \mathbf{Z}_0 + x_0 \cdot \mathbf{Z}_1)\big)$$

Application of the fixed point algorithm leads to:

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

Validity and realizability checking:

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

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

5.3 Approximate Version of Exact Model Checking for Incomplete Designs with One Combinational Black Box

The Black Box transformation into its corresponding Meta Black Box has a large impact on the number of variables needed, since the number of Meta Black Box outputs grows exponentially with the number of inputs of the original Black Box. Due to the expected complexity of this, the exact algorithm can be used only for instances with a little number of Black Box inputs.

We recall the reasoning we have done in Sect. 3 for the automaton shown in Fig. 5b (see Hypothesis 2 and 3): By giving two simple exemplary Black Box substitutions, we showed that there is a Black Box substitution so that $\varphi_2 = EX(EGy_0 \vee AGy_1)$ is satisfied as well as φ_2 is not satisfied for all Black Box substitutions. These two examples were the constant **0** function and the constant **1** function – functions without any inputs.

For the general case, we consider a Black Box and a 'simplified' Black Box for which some of the Black Box inputs have been removed. If there is a substitution of the simplified Black Box so that a certain property is satisfied, then there is a substitution of the original Black Box satisfying this property as well.

Applied to model checking, this means that if the exact symbolic model checking procedure for a circuit with a simplified Black Box returns that there is a substitution so that a CTL property φ is satisfied, there is a substitution for the non-simplified Black Box as well. In this way, we can prove realizability. Analogously, we can conclude from a non-valid result that there is a substitution of the Black Box not satisfying φ and thus falsify validity. So, by removing some of the Black Box inputs and performing exact symbolic model checking on the emerging incomplete circuit, we can retrieve sound results without the need to handle the complete Black Box truth table, as it would have been necessary when using exact symbolic model checking to verify the original incomplete circuit. We call this method *approximate version of exact symbolic model checking*.

Considering a Black Box input coming from a flipflop that has been previously extracted from a Black Box with bounded memory in order to create a combinational Black Box, it is easy to see that in case this Black Box input is removed, both the extracted flipflop and the according Black Box output can be removed too, since they are no longer connected to any logic and thus do not have any influence on the circuit behavior.

Our prototype implementation MIND starts with considering no Black Box inputs and adds them again until it is able to prove realizability, to falsify validity, or to make an exact statement by considering all Black Box inputs, which is equivalent to exact symbolic model checking.

The reasoning showed above can be applied to exact symbolic model checking, too: If the assumption of bounded memory reflects restrictions which really exist in a system, exact symbolic model checking will provide an exact proof of realizability or validity; otherwise, a Black Box with bounded memory can again be seen as a simplification of a Black Box with unbounded memory, making it possible to prove realizability and falsify validity for incomplete circuits containing one Black Box with unbounded memory.

5.4 Symbolic Model Checking for Multiple Black Boxes

Now, we consider a circuit containing more than one, say m Black Boxes. We assume that all flipflops are already extracted from the sequential Black Boxes as seen in Sect. 5.1, so that all Black Boxes are combinational. Let β^i $(1 \leq i \leq m)$ be the unknown function of the respective Black Box i. Each of these Black Boxes has an input function $\alpha^1, \ldots, \alpha^m$ and an output vector $\vec{Z}^1, \ldots, \vec{Z}^m$ (see Fig. 12). Furthermore we assume that the Black Boxes are

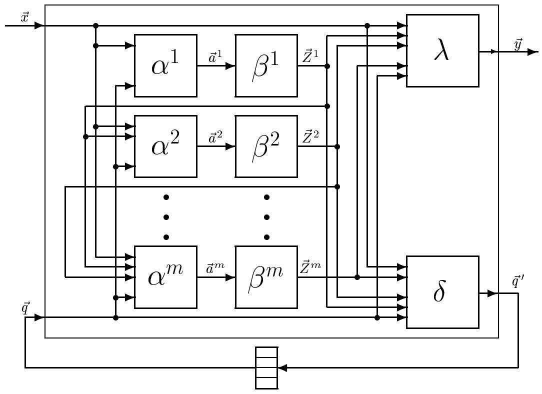


Figure 12: Mealy Automaton containing multiple Black Boxes

in topological order, meaning that each α^i only depends on $\vec{x}$, $\vec{q}$ and $\vec{Z}^0, \ldots, \vec{Z}^{i-1}$, but not $\vec{Z}^i, \ldots, \vec{Z}^m$.[4]

Each of these Black Boxes β^i can be transformed into a Meta Black Box $\boldsymbol{\beta}^i$ with readout function, which leads to a circuit with m combinational (Meta) Black Boxes without any inputs. It is easy to see that two Black Boxes without any input can be joined into one Black Box with no inputs and the sum of the two Black Boxes' outputs. Due to that, we can join all Meta Black Boxes $\boldsymbol{\beta}^i$ to one single Meta Black Box. Intuitively, this single Meta Black Box represents the aggregation of the truth tables of all original Black Boxes.

In the result of doing this, we receive an incomplete circuit with one single Meta Black Box, which can be handled by using exact symbolic model checking. Furthermore, we can again achieve an approximate version of this exact symbolic model checking procedure by simplifying some of the Black Boxes before they are converted to Meta Black Boxes.

5.5 Experimental Results

Our prototype implementation MIND, which has already been introduced in Sect. 4.3, is also able to perform exact symbolic model checking and its approximate version. For a given incomplete circuit and a CTL formula, MIND first checks whether it is possible to prove validity or to falsify realizability by using symbolic Z-model checking, symbolic Z_i-model checking and output consistent Z_i-model checking. In case this is not possible, MIND extracts the flipflops from the Black Boxes[5] and then performs the approximative version of the exact symbolic model checking procedure, starting with no Black Box inputs and adding them again until it is able to prove realizability or to falsify validity. If at the end all Black Box inputs had to be included again, then the method corresponds to exact symbolic model checking.

For our experiments, we used the same class of simple synchronous pipelined ALUs as in Sect. 4.3. Again, we compare a series of complete pipelined ALUs with 16 registers in the

[4] If there is no such order, there are replacements of the Black Boxes leading to non-combinational circuits.
[5] The number of flipflops in the Black Boxes result from upper bounds given by the user.

register file and varying word width to two incomplete pendants: For the first, the adder and the multiplier are substituted by Black Boxes and for the second, 12 of the 16 registers in the register file are masked out as well. Each Black Box's memory was bound to the number of flipflops the substituted circuit parts had, thus 0 for the adder and the multiplier (since they were combinational circuits) and the word width w for the registers in the register file.

All experiments were performed on an Intel Pentium4 2.6GHz with 1GB RAM and with a limited runtime of 12.000 seconds.

In a first experiment we checked the CTL formula $\varphi_3 = AG\big((EX)^3\mathbf{R} \equiv (AX)^3\mathbf{R}\big)$ from [3], which is true for the complete design. If some parts implementing ALU operations are masked out by Black Boxes, φ_3 remains valid for all possible replacements of the Black Boxes as already mentioned in Sect. 3 (for the incomplete pipelined ALU, in which a part of the register file has been removed, we only considered the remaining registers).

At first, we confined ourselves to proving realizability. We observed that our tool MIND was able to prove the realizability of φ_3 for the incomplete pipelined ALU, in which the adder and the multiplier have been masked out, up to a word width of 10 bit by using the approximative version of exact symbolic model checking considering no Black Box inputs. For the incomplete pipelined ALU, in which most of the register file has been replaced by Black Boxes as well, the realizability of φ_3 was proven up to a word width of 48 bit with the same method.

However, note that for proving also validity we have to consider all inputs of the Black Boxes, finally leading to an application of the exact symbolic model checking procedure. Thus, proving validity was only possible for small word width (up to 2 bits) with given resources.

In Tab. 4 we give the results for formula φ_3 and – with varying word width – both for the complete pipelined ALU and for proving realizability for the incomplete pipelined ALU. For each word width and each pipelined ALU, the table shows the number of BDD variables ('BDD vars'), the peak memory usage ('memory used'), the time spent while reordering the BDD variables ('RO time'), the peak number of BDD nodes and the overall time in CPU seconds.

Since a multiplier has 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 6 bit exceeds the time limit. In contrast to that, φ_3 can be proven to be realizable up to a word width of 10 bit for the incomplete pipelined ALUs without adder and multiplier.

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, making it possible to check φ_3 even up to a word width of 48 bit.

In a second experiment we considered a series of pipelined ALUs, in which adder, multiplier and register file were completely defined, but in which the control logic has been replaced by a Black Box. We checked $\varphi_1 = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$, which corresponds to formula (1) in [3] and which has already been used in Sect. 4.3. Obviously, φ_1 is realizable for the incomplete design, since φ_1 is satisfied if the Black Box is replaced by the original control logic, but it is not valid, since it is possible to replace the Black Box by a control logic that always executes another ALU function (e.g. an OR function), regardless of the command given at the inputs.

	No Black Boxes					**Adder and multiplier replaced by Black Boxes**					**Adder, multiplier and 12 registers replaced by Black Boxes**				
word width	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time	BDD vars	memory used	BDD nodes	RO time	time
2	115	9169636	86134	10.95	28.07	120	8760404	45958	16.35	63.63	96	8061332	30254	3.62	6.42
4	191	46662580	189340	59.92	201.41	200	35523796	186430	89.01	350.40	152	37550388	237256	85.27	146.03
6	267	51533348	247909	103.21	820.40	280	51468644	89242	38.14	498.32	208	27289972	44810	12.79	83.86
8	more than 12.000 sec.					360	54921796	266924	282.69	1286.33	264	51045412	220006	160.58	862.32
10						440	54531636	328496	359.05	2920.53	320	47202708	97833	85.23	347.20
12						more than 12.000 sec.					376	46820564	62226	31.47	268.34
16											488	44999300	109491	113.48	540.74
32											936	51248692	157856	329.93	3218.43
48											1384	52539412	142496	534.30	8809.51
64											more than 12.000 sec.				

Table 4: Pipelined ALU with 16 registers: Proving the realizability of $\varphi_3 = AG\big((EX)^3\mathbf{R} \equiv (AX)^3\mathbf{R}\big)$ using the approximate version of exact symbolic model checking

word width	BDD vars	memory used	BDD nodes	RO time	time
1	74	14897060	25332	1.29	5.11
2	112	10275588	70720	9.53	11.47
3	150	20327956	139241	34.06	43.80
4	188	83659412	470578	200.98	512.54
5	226	103940084	2537356	2108.86	40877.23
6	more than 60.000 sec.				

Table 5: Pipelined ALU with 16 registers, in which the control logic has been replaced by a Black Box: Proving the realizability and disproving the validity of $\varphi_1 = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ by using the approximate version of exact symbolic model checking

In Tab. 5 we give the results for the incomplete pipelined ALUs tested with φ_1. MIND was able to prove the realizability and disprove the validity up to a word width of 5 bits before exceeding an extended time limit of 60.000 seconds. Note that the pipelined ALUs considered in this series of experiments contained a combinational multiplier, which has a large impact on the number of BDD nodes and thus on computation time.

6 Conclusions and Future Work

In the first part of this paper, we introduced three approximate methods to realize symbolic model checking for incomplete designs. Our methods are able to provide sound results for falsifying realizability and for proving validity of incomplete designs (even if the Black Boxes lie inside the cone of influence for the considered CTL formula).

Additionally, we introduced a concept for exact symbolic model checking of incomplete designs containing several Black Boxes with bounded memory and developed an approximate version of this method trading off accuracy and computational resources. This approximate version is still able to provide sound results for proving realizability and for falsifying validity of incomplete designs including Black Boxes with bounded memory (potentially even including Black Boxes with unbounded memory).

Experimental results using our prototype implementation MIND proved that the need for computational resources (memory and time) could be substantially decreased by masking complex parts of a design and by using model checking for the resulting incomplete design. The increase in efficiency was obtained while still providing sound and useful results. Further, we were able to handle circuits with masked control logic, proving the realizability and disproving the validity of the given property.

At the moment we are working on further improvements concerning the accuracy of our approximate symbolic model checking methods.

References

[1] 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*, 8(2):244–263, 1986.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] 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*, 98(2):142–170, 1992.

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

[5] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.

[6] The VIS Group. VIS: A system for verification and synthesis. In *Computer Aided Verification*, volume 1102 of *LNCS*, pages 428–432. Springer Verlag, 1996.

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

[8] K. L. McMillan. *The SMV language*. Cadence Berkeley Labs.

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

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

[11] C. Scholl and B. Becker. Checking equivalence for partial implementations. In *Design Automation Conf.*, pages 238–243, 2001.

[12] Michael Huth, Radha Jagadeesan, and David Schmidt. Modal transition systems: A foundation for three-valued program analysis. In Sands D., editor, *Proceedings of European Symposium on Programming*, number 2028 in Lecture Notes in Computer Science, pages 155+. Springer, April 2001.

[13] J.R. Burch and D.L. Dill. Automatic verification of microprocessor control. In *Computer Aided Verification*, volume 818 of *LNCS*, pages 68–80. Springer Verlag, 1994.

[14] K. Sajid, A. Goel, H. Zhou, A. Aziz, and V. Singhal. BDD-based procedures for a theory of equality with uninterpreted functions. In *Computer Aided Verification*, volume 1447 of *LNCS*, pages 244–255. Springer Verlag, 1998.

[15] S. Berezin, A. Biere, E.M. Clarke, and Y. Zhu. Combining symbolic model checking with uninterpreted functions for out-of-order processor verification. In *Int'l Conf. on Formal Methods in CAD*, pages 369–386, 1998.

[16] R.E. Bryant, S. German, and M.N. Velev. Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic. *ACM Transactions on Computational Logic*, 2(1):1–41, 2001.

[17] 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.*, pages 263–269, 2000.

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

[19] C. Scholl and B. Becker. Checking equivalence for partial implementations. Technical Report 145, Albert-Ludwigs-University, Freiburg, October 2000.

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

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

[22] R. Hojati, S.C. Krishnan, and R.K. Brayton. Early quantification and partitioned transition relations. In *Int'l Conf. on Comp. Design*, pages 12–19, 1996.