Approximate Symbolic Model Checking for Incomplete Designs

Tobias Nopper
Institute of Computer Science
Albert-Ludwigs-University Freiburg
D-79110 Freiburg im Breisgau, Germany
nopper@informatik.uni-freiburg.de

Christoph Scholl
Institute of Computer Science
Albert-Ludwigs-University Freiburg
D-79110 Freiburg im Breisgau, Germany
scholl@informatik.uni-freiburg.de

ABSTRACT
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. 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 lead 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 approach two potentially interesting questions, whether it is still possible to replace the Black Boxes by circuit implementations, so that a given model checking property is satisfied (‘realizability’) and 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 modelling 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 fulfills a given property?’) or validity (‘is a given property fulfilled for all replacements of the Black Boxes?’). This approach is even not able to provide approximate solutions to 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. 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. First experimental results given in Section 5 prove effectiveness and feasibility of the 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 $\phi$. During one run of symbolic model checking we compute both underapproximations and overapproximations of the states satisfying $\phi$ and we will use them to provide approximate answers for realizability and validity.

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 [12], where a validity problem is solved under the assumption that both specification and implementation contain the same Uninterpreted Functions. Whereas in [12, 13, 14, 15] 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 $CTL$. The paper is structured as follows: After giving a brief review of symbolic model checking and of representations for incomplete designs in Section 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 Section 3. In Section 4, we will introduce several algorithms capable of performing sound and approximate symbolic model checking for incomplete circuits. Finally we give a series of experimental results demonstrating the effectiveness and feasibility of the presented methods in Section 5 and we conclude the paper in Section 6.

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 to and 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 $M := (B^{[\delta]}, B^{[\gamma]}, B^{[\gamma]}, \delta, \lambda, q^0)$ with state set $B^{[\delta]}$, the set of inputs $B^{[\gamma]}$, the set of outputs $B^{[\gamma]}$, transition function $\delta: B^{[\delta]} \times B^{[\gamma]} \rightarrow B^{[\delta]}$, output function $\lambda: B^{[\delta]} \times B^{[\gamma]} \rightarrow B^{[\gamma]}$, and initial state $q^0 \in B^{[\delta]}$. In the following we will use $\vec{x} = (x_0, \ldots, x_{n-1})$ with $n = |\vec{x}|$ for vectors of input variables, $\vec{y}$ for vectors of output variables, $q$ for current state variables and $q^0$ for next state variables.

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 $\text{struct}(M) := (S, R, L)$ with

$$S := B^{[\delta]} \times B^{[\gamma]}$$

$$R = \{ (\vec{q}, \vec{x}, \vec{q}', \vec{x}') \in B^{[\delta]} \times B^{[\gamma]} \times B^{[\delta]} \times B^{[\gamma]} : \delta(\vec{q}, \vec{x}) = \vec{q}' \land \chi(\vec{q}, \vec{x}) \}$$

$$L((\vec{q}, \vec{e})) := \{ x_i \mid e_i = 1 \} \cup \{ y_i \mid \lambda(\vec{q}, \vec{e}) = 1 \}.$$ 

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

$$s \models \varphi \land \psi \iff s \models \varphi \land s \models \psi$$

$$s \models \neg \varphi \iff \neg s \models \varphi$$

$$s \models \varphi_1 \lor \varphi_2 \iff s \models \varphi_1 \lor s \models \varphi_2$$

$$s \models \varphi \rightarrow \psi \iff s \models \varphi \land s' \models \psi$$

$$s \models E\varphi \iff \exists s' \in S: R(s, s') \land s' \models \varphi$$

$$s \models EG\varphi \iff s \models \varphi \land \exists s' \in S: R(s, s') \land s' \models EG\varphi$$

and

$$s \models E\varphi_1 U\varphi_2 \iff s \models \varphi_2 \lor (s' \in S: R(s, s')$$

$$\quad \land s' \models E\varphi_1 U\varphi_2)$$

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

In symbolic model checking, sets of states are represented by characteristic functions, which are in turn represented by BDDs. Let $\chi_{\text{Sat}}(\varphi)$ be the set of states of $\text{struct}(M)$ which satisfy formula $\varphi$ and let $\chi_{\text{Sat}}(\varphi)$ be its characteristic function, then $\chi_{\text{Sat}}(\varphi)$ can be computed recursively based on the characteristic function $\chi_R(\vec{q}, \vec{x}, \vec{q}')$ of the transition relation $R$:

$$\chi_{\text{Sat}}(x_i)(\vec{q}, \vec{x}) := x_i$$

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

$$\chi_{\text{Sat}}(\neg \varphi)(\vec{q}, \vec{x}) := \overline{\chi_{\text{Sat}}(\varphi)}(\vec{q}, \vec{x})$$

$$\chi_{\text{Sat}}(\varphi_1 \lor \varphi_2)(\vec{q}, \vec{x}) := \chi_{\text{Sat}}(\varphi_1)(\vec{q}, \vec{x}) + \chi_{\text{Sat}}(\varphi_2)(\vec{q}, \vec{x})$$

$$\chi_{\text{Sat}}(EX\varphi)(\vec{q}, \vec{x}) := \chi_{\text{EX}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x})$$

$$\chi_{\text{Sat}}(EG\varphi)(\vec{q}, \vec{x}) := \chi_{\text{EG}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x})$$

$$\chi_{\text{Sat}}(E\varphi_1 U\varphi_2)(\vec{q}, \vec{x}) := \chi_{\text{EV}}(\chi_{\text{Sat}}(\varphi_1) \land \chi_{\text{Sat}}(\varphi_2))(\vec{q}, \vec{x})$$

with

$$\chi_{\text{EX}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x}) := \chi_{\text{EX}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x}) \land \exists \vec{q}' \in R(\vec{q}, \vec{x}) (\chi_{\text{Sat}}(\varphi)(\vec{q}', \vec{x}') \land \chi_{\text{EX}}(\chi_{\text{Sat}}(\varphi))(\vec{q}', \vec{x}'))$$

and

$$\chi_{\text{EG}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x}) := \chi_{\text{EG}}(\chi_{\text{Sat}}(\varphi))(\vec{q}, \vec{x}) \land \exists \vec{q}' \in R(\vec{q}, \vec{x}) (\chi_{\text{Sat}}(\varphi)(\vec{q}', \vec{x}') \land \chi_{\text{EG}}(\chi_{\text{Sat}}(\varphi))(\vec{q}', \vec{x}'))$$

$\chi_{\text{EG}}$ and $\chi_{\text{EU}}$ can be evaluated by the fixed point iteration algorithms shown in Fig. 1.

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

$$M \models \varphi \iff \forall \vec{x} \in B^{[\gamma]} : \text{struct}(M), (q^0, \vec{x}) \models \varphi$$

$$\quad \iff \forall \vec{x} \in B^{[\gamma]} : \chi_{\text{Sat}}(\varphi)(q^0, \vec{x}) = 1$$

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 Section 4.

Unknown parts of the design are combined into so-called ‘Black Boxes’ (see Fig. 2a for a combinational example with one Black Box).

Figure 1: Fixed point iteration algorithms

a) Fixed point iteration for $EF$

```c
\text{ch}_E(\vec{x} \& \vec{y}) = \{ \text{odd} := 0; \text{new} := \vec{x}; \text{while} (\text{odd} \neq \text{new}) \{ \text{odd} := \vec{y}; \text{new} := \vec{x} \& \text{ch}_E(\vec{y} \& \text{new}); \} \text{return new; } \}
```

b) Fixed point iteration for $EU$

```c
\text{ch}_U(\vec{x} \& \vec{y}) = \{ \text{odd} := 0; \text{new} := \vec{x}; \text{while} (\text{odd} \neq \text{new}) \{ \text{odd} := \vec{y}; \text{new} := \vec{x} \& \text{ch}_U(\vec{y} \& \text{new}); \} \text{return new; } \}
```
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 $\varphi$? If this is true, then the property $\varphi$ is called realizable for the incomplete design. The corresponding decision problem is called realizability problem.

2. Is a CTL formula $\varphi$ satisfied for all possible replacements of the Black Boxes? If this is the case, then $\varphi$ 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 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. 3]. In [3], Burch et al. showed by symbolic model checking that (among other CTL formulas) the following formulas are satisfied for the pipelined ALU:

\begin{align}
AG((EX)^3 R &\equiv (AX)^3 R) \\
AG((EX)^2 R &\equiv (AX)^2 R)
\end{align}

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. However, it is clear that formula (1) is satisfied.

The formulas essentially say that the content of the register file $R$ two (resp. three) clock cycles in the future is uniquely determined by the current state of the system.

Using VIS, the verification already fails for formula (1) — this is due to a slightly different modelling of automata by Kripke structures in VIS and SMV.
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 $\varphi_2$ is satisfied if the Black Box is substituted with the constant 1 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. 4b in combination with $\varphi_3 = \neg \varphi_1 = \neg AG(AXy_0 \lor AX \neg y_0)$. 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 $\varphi_3$ is satisfied. However, since property $\varphi_3$ is the negation of property $\varphi_1$ which has been proven to be valid when considering the first hypothesis, it is quite obvious that $\varphi_3$ 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. 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. 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 $\varphi$ and then checks whether all initial states are included in this set. If so, the circuit satisfies $\varphi$.

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 $\varphi$. In contrast to conventional model checking we will consider two sets instead of $Sat(\varphi)$: The first set is called $Sat^{ex}_R(\varphi)$ and it contains all states, for which there is at least one Black Box replacement so that $\varphi$ is satisfied. To obtain $Sat^{ex}_R(\varphi)$ 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 $Sat^{ex}_R(\varphi)$ as the union of all these sets $Sat^R(\varphi)$. The second set is called $Sat^{\neg ex}_R(\varphi)$ and it contains all states, for which $\varphi$ is satisfied for all Black Box replacements. Conceptually, $Sat^{\neg ex}_R(\varphi)$ could be computed as an intersection of all sets $Sat^R(\varphi)$ obtained for all possible replacements $R$ of the Black Boxes.

Given $Sat^{ex}_R(\varphi)$ and $Sat^{\neg ex}_R(\varphi)$, it is easy to prove validity and to falsify realizability for the incomplete circuit: If all initial states are included in $Sat^{ex}_R(\varphi)$, then all initial states are included in $Sat^R(\varphi)$ for each replacement.
4.2 Approximations

For reasons of efficiency we will not compute exact sets $\text{Sat}_E^\Delta(\varphi)$ and $\text{Sat}_A^\Delta(\varphi)$. Instead we will compute approximations $\text{Sat}_E(\varphi)$ and $\text{Sat}_A(\varphi)$ of these sets. To be more precise we will compute overapproximations $\text{Sat}_E(\varphi) \supseteq \text{Sat}_E^\Delta(\varphi)$ of $\text{Sat}_E^\Delta(\varphi)$ and underapproximations $\text{Sat}_A(\varphi) \subseteq \text{Sat}_A^\Delta(\varphi)$ of $\text{Sat}_A^\Delta(\varphi)$.

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

Approximations of $\text{Sat}_E(\varphi)$ and $\text{Sat}_A(\varphi)$ 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 $\delta$ and the output function $\lambda$ (see Figure 5). For this reason there are two types of transitions for the automaton:

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

We will work with two types of approximations of the transition relation $\chi_R(\vec{q}, \vec{x}, \vec{q'})$. An underapproximation $\chi_{R_\lambda}(\vec{q}, \vec{x}, \vec{q'})$ will only contain fixed transitions and an overapproximation $\chi_{R_\delta}(\vec{q}, \vec{x}, \vec{q'})$ 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 $\text{Sat}(y_i)$ in which the output value $y_i$ of $\lambda_i$ is true:

- an underapproximation $\text{Sat}_A(y_i)$ contains only states in which $y_i$ is true independently from the replacements of the Black Boxes and
- an overapproximation $\text{Sat}_E(y_i)$ contains at least all states in which $y_i$ may be true for some replacement of the Black Boxes.

Based on these approximations $\chi_{R_\lambda}, \chi_{R_\delta}, \text{Sat}_A(y_i)$, and $\text{Sat}_E(y_i)$ we will compute the underapproximations $\text{Sat}_A(\varphi)$ and overapproximations $\text{Sat}_E(\varphi)$ mentioned above for arbitrary CTL formulas $\varphi$.

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 $\delta$ and $\lambda$ as described in Section 2.

4.2.1 Symbolic $Z$-Model Checking

We apply symbolic $(0, 1, X)$-simulation (see Section 2) for computing $\delta$ and $\lambda$. 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 $\delta_i(\vec{q}, \vec{x}, Z)$.

Output functions

If $\lambda \models |\vec{q'} - \vec{q}|, \vec{x} - \vec{x}, Z = 1$ for some state $\vec{q'}$, then we know that $\lambda_i$ is 1 in this state independently from the replacement of the Black Boxes, so we include $\vec{q'}, \vec{x}$ into both $\text{Sat}_A(y_i)$ and $\text{Sat}_E(y_i)$. If $\lambda \models |\vec{q'} - \vec{q}|, \vec{x} = Z = \text{Sat}_E(y_i)$, then the output $\lambda_i$ may or may not be equal to 1 and thus, we include $\vec{q'}, \vec{x}$ into $\text{Sat}_E(y_i)$, but not into $\text{Sat}_A(y_i)$. This leads to the following symbolic representations:

$$
\chi_{\text{Sat}_A(y_i)}(\vec{q}, \vec{x}, Z) = \forall Z(\lambda_i(\vec{q}, \vec{x}, Z)),
\chi_{\text{Sat}_E(y_i)}(\vec{q}, \vec{x}, Z) = \exists Z(\lambda_i(\vec{q}, \vec{x}, Z)).
$$

Transition functions

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

$$
\chi_{R_\lambda}(\vec{q}, \vec{x}, \vec{q'}) = \left( \prod_{i=0}^{\vec{q}^1-1} \forall Z(\delta_i(\vec{q'}, \vec{x}, Z) \equiv q_i') \right)
$$

and for $\chi_{R_\delta}$, representing at least all possible transitions we obtain

$$
\chi_{R_\delta}(\vec{q}, \vec{x}, \vec{q'}) = \left( \prod_{i=0}^{\vec{q}^1-1} \exists Z(\delta_i(\vec{q'}, \vec{x}, Z) \equiv q_i') \right).
$$

Note that $\chi_{R_\lambda}$ defined in this way underapproximates the set of all fixed transitions due to well-known deficiencies of $(0, 1, X)$-simulation [11] and $\chi_{R_\delta}$ overapproximates the set of all possible transitions (the same is true for $\chi_{\text{Sat}_A(y_i)}$ and $\chi_{\text{Sat}_E(y_i)}$, respectively).

In order to compute $\text{Sat}_E(\varphi)$ and $\text{Sat}_E(\varphi)$ recursively for arbitrary CTL formulas we need rules to evaluate operators $EX, \neg, \lor, \lor, EG$ and $EU$.
Figure 6: Evaluation of $Sat_A(\overline{EX}\psi)$ and $Sat_E(\overline{EX}\psi)$

Computing $Sat_A(\overline{EX}\psi)$ and $Sat_E(\overline{EX}\psi)$

Given $Sat_A(\psi)$, the set of states which definitely satisfy $\psi$ for all Black Box replacements, we include into $Sat_A(\overline{EX}\psi)$ 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 $Sat_E(\overline{EX}\psi)$ which have a possible transition to a state in $Sat_E(\psi)$. Fig. 6 illustrates the sets. Thus, we have

$$\chi_{Sat_A(\overline{EX}\psi)}(\vec{q}, \vec{x}) = \exists\vec{q}' \exists\vec{x}' (\chi_{Sat_A(\psi)}(\vec{q}', \vec{x}') \cdot (\chi_{Sat_A(\overline{EX}\psi)}(\vec{q}'', \vec{x}''))(\vec{q}'', \vec{x}''))$$

and

$$\chi_{Sat_E(\overline{EX}\psi)}(\vec{q}, \vec{x}) = \exists\vec{q}' \exists\vec{x}' (\chi_{Sat_E(\psi)}(\vec{q}', \vec{x}') \cdot (\chi_{Sat_E(\overline{EX}\psi)}(\vec{q}'', \vec{x}''))(\vec{q}'', \vec{x}''))$$

Computing $Sat_A(\neg\psi)$ and $Sat_E(\neg\psi)$

$Sat_A(\neg\psi)$ is an overapproximation of all states in which $\psi$ may be satisfied for some Black Box replacement. Thus, we do know that for an arbitrary state in $B^{[q]} \times B^{[\vec{q}]} \setminus Sat_A(\psi)$ there is no Black Box replacement so that $\psi$ 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 $B^{[q]} \times B^{[\vec{q}]} \setminus Sat_E(\psi)$ as an underapproximation $Sat_A(\neg\psi)$. Since an analogous argument holds for $Sat_A(\neg\psi)$ and $Sat_E(\psi)$ we define

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

and

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

Evaluating $\forall$, $EG$ and $EU$

It is easy to see that $\chi_{Sat_A(\forall\varphi)}(\vec{q}, \vec{x}) = \chi_{Sat_A(\forall\varphi)}(\vec{q}, \vec{x}) \vee \chi_{Sat_A(\exists\varphi)}(\vec{q}, \vec{x}) \vee \chi_{Sat_E(\forall\varphi)}(\vec{q}, \vec{x}) \vee \chi_{Sat_E(\exists\varphi)}(\vec{q}, \vec{x})$. Moreover, $\varphi = EG\psi$ and $\varphi = EU\psi$ can be evaluated by standard fixed point iterations according to Figures 1a and 1b based on the evaluation of $EX$ defined above (two separate fixed point iterations for $Sat_A$ and $Sat_E$).

Alternatively, 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 $\chi_R = \chi_{Sat_A(\varphi)}(\vec{q}, \vec{x}) + e \cdot \chi_{Sat_E(\varphi)}(\vec{q}, \vec{x})$, we can easily combine the two computations of $\chi_{Sat_A(\varphi)}$ and $\chi_{Sat_E(\varphi)}$ into one computation for $\chi_{Sat(\varphi)} = \chi_{Sat_A(\varphi)} + e \cdot \chi_{Sat_E(\varphi)}$. More details can be found in [19].

Example

Again, we consider the incomplete circuit shown in Figure 4a. 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 $\varphi := AG(\neg y_0 \lor \neg y_1)$. 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.

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 $\delta_i(\vec{q}, \vec{x}, \vec{Z})$ will now depend on a vector $\vec{Z}$ of additional variables. As in the previous section, we include a state $(\vec{q}^{mx}, \vec{x}^{mx}) \in B^{[q]} \times B^{[\vec{x}]}$ into $Sat_A(y_i)$ iff $\lambda_i[1]_{\vec{q}^{mx}, \vec{x}^{mx}, \vec{Z}^{mx}} = 1$ and we include it into $Sat_E(y_i)$ iff $\lambda_i[1]_{\vec{q}^{mx}, \vec{x}^{mx}, \vec{Z}^{mx}} = 1$ or $\lambda_i[0]_{\vec{q}^{mx}, \vec{x}^{mx}, \vec{Z}^{mx}}$ 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 $Sat_A(\varphi)$ 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

$$\chi_{Sat}(\vec{q}, \vec{x}, \vec{q}') = \chi_{Sat}(\vec{q}, \vec{x}) = \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x})$$

(by equation (4))

$$\chi_{Sat}(\vec{q}, \vec{x}, \vec{q}') = \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x}) \vee \chi_{Sat}(\vec{q}, \vec{x})$$

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. 4a (with flip flop initialized to 0) to motivate the need for an improvement. Consider the CTL formula $EF(y_1 \land \neg y_1)$. 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.
Due to that, how the adder and multiplier function are implemented, the property is true for the complete design, independently of second and third register in the register file. This property holds true for the complete pipelined ALUs tested with incomplete pipelined ALUs with varying word width to two incomplete pendants: F or our experiments we used a class of simple synchronous pipelined ALUs similar to the ones presented in [3] (see also section 3, Fig. 3). In contrast to [3], our pipelined ALU contains a combinational multiplier. 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 of one single ALU – the multiplier and the adder – and most of the register file has been replaced by Black Boxes as already mentioned in Section 3 (for the incomplete pipelined ALU, in which a part of the register file has been removed, we only checked the remaining registers). Note that in this example, the Black Boxes lie inside the cone of influence for the CTL formula.

In Tab. 2 we give the results for both complete and incomplete pipelined ALUs tested with \( \varphi_3 \). In this example, symbolic Z-model checking and symbolic Z\(_i\)-model checking were not able to prove the validity of \( \varphi_2 \). 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 exact \( Z_i \)-model checking of the incomplete pipelined ALUs outperforms the conventional model checking of the complete version – for the same reasons as given above.

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. 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.

For our experiments we used a class of simple synchronous pipelined ALUs similar to the ones presented in [3] (see also section 3, Fig. 3). In contrast to [3], our pipelined ALU contains a combinational multiplier. 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 of one single ALU – the multiplier and the adder – and most of the register file has been replaced by Black Boxes as already mentioned in Section 3 (for the incomplete pipelined ALU, in which a part of the register file has been removed, we only checked the remaining registers). Note that in this example, the Black Boxes lie inside the cone of influence for the CTL formula.

In Tab. 2 we give the results for both complete and incomplete pipelined ALUs tested with \( \varphi_3 \). In this example, symbolic Z-model checking and symbolic Z\(_i\)-model checking were not able to prove the validity of \( \varphi_2 \). 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 exact \( Z_i\)-model checking of the incomplete pipelined ALUs outperforms the conventional model checking of the complete version – for the same reasons as given above.

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.

6. CONCLUSIONS AND FUTURE WORK

We introduced three approximate methods to realize symbolic model checking for incomplete designs. Our methods are able to provide sound results for falsify-
Table 1: Pipelined ALU with 16 registers: Proving the validity of $\varphi_1 = AG(\mathcal{R}_2 = 0 \oplus \mathcal{R}_1 \rightarrow (\mathcal{R}_0 \oplus \mathcal{R}_1) \equiv (AX^3 \mathcal{R}_1))$ using Symbolic $Z$-Model Checking

<table>
<thead>
<tr>
<th>word width</th>
<th>BDD vars</th>
<th>memory used</th>
<th>BDD nodes</th>
<th>RO time</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>117</td>
<td>89092957</td>
<td>1372</td>
<td>6.52</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>193</td>
<td>5557560</td>
<td>1324</td>
<td>3.36</td>
<td>4.33</td>
</tr>
<tr>
<td>8</td>
<td>345</td>
<td>8137772</td>
<td>1474</td>
<td>10.93</td>
<td>13.21</td>
</tr>
<tr>
<td>12</td>
<td>497</td>
<td>97710484</td>
<td>10370</td>
<td>226.45</td>
<td>387.51</td>
</tr>
<tr>
<td>16</td>
<td>682</td>
<td>51202224</td>
<td>16000</td>
<td>85.75</td>
<td>130.31</td>
</tr>
<tr>
<td>more than 12,000 sec</td>
<td>1522</td>
<td>44936409</td>
<td>22833</td>
<td>586.19</td>
<td>755.10</td>
</tr>
<tr>
<td>64</td>
<td>2602</td>
<td>60688692</td>
<td>01678</td>
<td>382.05</td>
<td>1823.80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>word width</th>
<th>BDD vars</th>
<th>memory used</th>
<th>BDD nodes</th>
<th>RO time</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>63609</td>
<td>1331</td>
<td>2.45</td>
<td>2.45</td>
</tr>
<tr>
<td>4</td>
<td>193</td>
<td>14478612</td>
<td>8947</td>
<td>18.52</td>
<td>31.40</td>
</tr>
<tr>
<td>8</td>
<td>345</td>
<td>14346869</td>
<td>10140</td>
<td>76.62</td>
<td>82.20</td>
</tr>
<tr>
<td>12</td>
<td>497</td>
<td>14257588</td>
<td>239538</td>
<td>243.13</td>
<td>258.77</td>
</tr>
<tr>
<td>16</td>
<td>682</td>
<td>14407260</td>
<td>142994</td>
<td>1429.94</td>
<td>1429.94</td>
</tr>
<tr>
<td>more than 12,000 sec</td>
<td>1251</td>
<td>71885004</td>
<td>2591074</td>
<td>10684.89</td>
<td>10688.37</td>
</tr>
<tr>
<td>64</td>
<td>2931</td>
<td>613970086</td>
<td>336098</td>
<td>794.05</td>
<td>799.98</td>
</tr>
</tbody>
</table>

Table 2: Pipelined ALU with 16 registers: Proving the validity of $\varphi_2 = AG((EX)^2 \mathcal{R} \equiv (AX)^2 \mathcal{R})$ using Output Consistent $Z$-Model Checking

<table>
<thead>
<tr>
<th>word width</th>
<th>BDD vars</th>
<th>memory used</th>
<th>BDD nodes</th>
<th>RO time</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>117</td>
<td>89092957</td>
<td>1372</td>
<td>6.52</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>193</td>
<td>5557560</td>
<td>1324</td>
<td>3.36</td>
<td>4.33</td>
</tr>
<tr>
<td>8</td>
<td>345</td>
<td>8137772</td>
<td>1474</td>
<td>10.93</td>
<td>13.21</td>
</tr>
<tr>
<td>12</td>
<td>497</td>
<td>97710484</td>
<td>10370</td>
<td>226.45</td>
<td>387.51</td>
</tr>
<tr>
<td>16</td>
<td>682</td>
<td>51202224</td>
<td>16000</td>
<td>85.75</td>
<td>130.31</td>
</tr>
<tr>
<td>more than 12,000 sec</td>
<td>1522</td>
<td>44936409</td>
<td>22833</td>
<td>586.19</td>
<td>755.10</td>
</tr>
<tr>
<td>64</td>
<td>2602</td>
<td>60688692</td>
<td>01678</td>
<td>382.05</td>
<td>1823.80</td>
</tr>
</tbody>
</table>

7. REFERENCES


