Preprint from Proceedings of IWLS2004, Temecula, USA, June 2004

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 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. 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 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.

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

a) Fixed point iteration for *EG*

$\chi_{EG}(\chi_X)$ {
  $old := 1$;
  $new := \chi_X$;
  while ($old \neq new$) {
    $old := new$;
    $new := \chi_X \cdot \chi_{EX}(old)$;
  }
  return new;
}

b) Fixed point iteration for *EU*

$\chi_{EU}(\chi_X, \chi_Y)$ {
  $old := 0$;
  $new := \chi_Y$;
  while ($old \neq new$) {
    $old := new$;
    $new := \chi_Y + (\chi_X \cdot \chi_{EX}(old))$;
  }
  return new;
}

Figure 1: Fixed point iteration algorithms

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 and to a formula of a temporal logic (in our case CTL (Computation Tree Logic)) on the other hand.

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

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

$$S := \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|}$$

$$R := \big\{\big((\vec{q},\vec{x}),(\vec{q}\,',\vec{x}\,')\big)\big|\vec{q},\vec{q}\,'\in\mathbb{B}^{|\vec{q}|},\vec{x},\vec{x}\,'\in\mathbb{B}^{|\vec{x}|},\ \delta(\vec{q},\vec{x})=\vec{q}\,'\big\}$$

$$L\big((\vec{q},\vec{\epsilon})\big) := \big\{x_i\big|\epsilon_i = 1\big\} \cup \big\{y_i\big|\lambda_i(\vec{q},\vec{\epsilon}) = 1\big\}.$$

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:

$$\begin{aligned}
&s \models \varphi;\ \varphi \in V && \iff \varphi \in L(s) \quad (V\!=\!\text{set of atomic prop.})\\
&s \models \neg\varphi && \iff s \not\models \varphi\\
&s \models (\varphi_1 \vee \varphi_2) && \iff s \models \varphi_1 \text{ or } s \models \varphi_2\\
&s \models EX\varphi && \iff \exists s' \in S\colon R(s,s') \text{ and } s' \models \varphi\\
&s \models EG\varphi && \iff s \models \varphi \text{ and } \exists s' \in S\colon R(s,s')\\
& && \qquad\quad \text{and } s' \models EG\varphi\\
&s \models E\varphi_1 U \varphi_2 && \iff s \models \varphi_2 \text{ or } \big(\exists s' \in S\colon R(s,s')\\
& && \qquad\quad \text{and } s \models \varphi_1 \text{ and } s' \models E\varphi_1 U \varphi_2\big)
\end{aligned}$$

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:

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

with

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

χ_{EG} and χ_{EU} can be evaluated by the fixed point iteration algorithms shown in Fig. 1.

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 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).

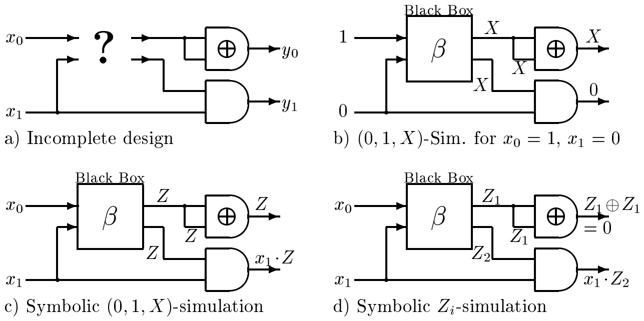

Figure 2: Incomplete design

For simulating the circuit wrt. some input vector we can make use of the ternary $(0, 1, X)$-logic [16, 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 [17] (see Figure 2b). If the value of some primary output is X, we do not know the value due to the unknown behaviour 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 [18, 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

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

This modified version of symbolic simulation is called *symbolic (0,1,X)-simulation*, see Fig. 2c 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 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]. Figure 2d shows an example for symbolic Z_i-simulation. (Note that the first output can now be shown to be constant 0.)

In Section 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 Section 4 we will present methods realizing approximate symbolic model checking for incomplete designs. We will consider two types of questions:

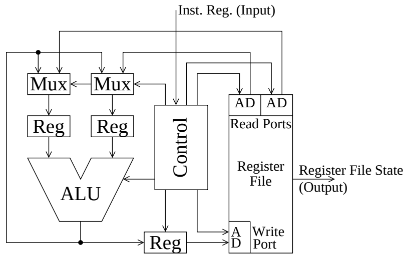

Figure 3: Pipelined ALU

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 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:[1]

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

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

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 modelled by non-deterministic signals. In this situation SMV provides the result that formula (2) is not satisfied.[2] However, it is clear

[1]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.

[2]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.

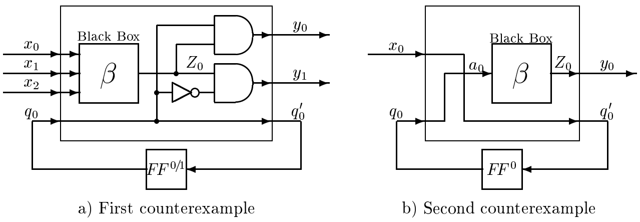

Figure 4: Counterexamples

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

Fig. 4b shows a counterexample for this hypothesis: If we substitute the Black Box output by a 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. 4a 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. 4a 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. 4b 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. 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 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 $Sat_E^{\mathrm{ex}}(\varphi)$ and it contains all states, for which *there is* at least one Black Box replacement so that φ is satisfied. To obtain $Sat_E^{\mathrm{ex}}(\varphi)$ 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 $Sat_E^{\mathrm{ex}}(\varphi)$ as the union of all these sets $Sat^R(\varphi)$. The second set is called $Sat_A^{\mathrm{ex}}(\varphi)$ and it contains all states, for which φ is satisfied for *all* Black Box replacements. Conceptually, $Sat_A^{\mathrm{ex}}(\varphi)$ could be computed as an intersection of all sets $Sat^R(\varphi)$ obtained for all possible replacements R of the Black Boxes.

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

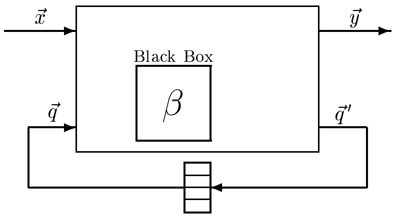

Figure 5: Mealy automaton with Black Box

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 $Sat_E^{\mathrm{ex}}(\varphi)$, 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").

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 Figure 5). 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.

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 both $Sat_A(y_i)$ 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:

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

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

$$\chi_{R_A}(\vec{q}, \vec{x}, \vec{q}') = \Big(\prod_{i=0}^{|\vec{q}|-1} \forall Z\big(\delta_i(\vec{q}, \vec{x}, Z) \equiv q_i'\big)\Big) \qquad (3)$$

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

$$\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}') = \Big(\prod_{i=0}^{|\vec{q}|-1} \exists Z\big(\delta_i(\vec{q}, \vec{x}, Z) \equiv q_i'\big)\Big). \qquad (4)$$

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.

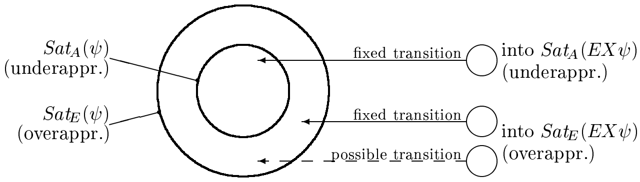

Figure 6: Evaluation of $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$

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. 6 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

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

Evaluating $\vee$, EG and EU

It is easy to see that $\chi_{Sat_A(\varphi_1\vee\varphi_2)}(\vec{q},\vec{x}) = \chi_{Sat_A(\varphi_1)}(\vec{q},\vec{x})\vee\chi_{Sat_A(\varphi_2)}(\vec{q},\vec{x})$ and $\chi_{Sat_E(\varphi_1\vee\varphi_2)}(\vec{q},\vec{x}) = \chi_{Sat_E(\varphi_1)}(\vec{q},\vec{x})\vee\chi_{Sat_E(\varphi_2)}(\vec{q},\vec{x})$. Moreover, $\varphi = EG\psi$ and $\varphi = E\psi_1 U\psi_2$ 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).

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 $\chi_{Sat(\varphi)} = \chi_{Sat_A(\varphi)} + e\cdot\chi_{Sat_E(\varphi)}$. 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 $\varphi := AG(\neg y_0 \vee \neg y_1)$. 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.

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

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

(equation (4)) by

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

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. 4a (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.

Detailed information on modifications which have to be made for this version of the model checking procedure is omitted due to lack of space. It can be found in [19].

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 neccessary – 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 in the register file are masked out as well.

All experiments were performed on an Athlon Thunderbird 800MHz with 512MB RAM and with a limited runtime of 12.000 seconds.

In a first experiment we checked the CTL formula $\varphi_1 = AG\big(''\mathtt{R}_2{:=}\mathtt{R}_0{\oplus}\mathtt{R}_1'' \rightarrow \big((\mathbf{R}_0 \oplus \mathbf{R}_1) \equiv (AX)^3\mathbf{R}_2\big)\big)$ 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 present state ($\mathbf{R}_0$, $\mathbf{R}_1$ and $\mathbf{R}_2$ are the respective first, second and third register in the register file). This property is true for the complete design, independently of how the adder and multiplier function are implemented. Due to that, φ_1 is also satisfied for all possible Black Box replacements in the incomplete pipelined ALUs.

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 ('memory used'), 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 valid up to a word width of 32 bit.

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 validity 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.

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 checked the CTL formula $\varphi_2 = AG\big((EX)^2\mathbf{R} \equiv (AX)^2\mathbf{R}\big)$ from [3], which is true for the complete design. If some parts implementing ALU operations are masked out by Black Boxes, φ_2 remains valid for all possible replacements of the 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 φ_2. In this example, symbolic Z-model checking and symbolic Z_i-model checking were not able to prove the validity of φ_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 particulary 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-

	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	117	5630804	13134	2.15	3.43	119	7873380	21331	2.95	5.58	71	4928468	1738	0.42	0.48
4	193	14475812	89447	18.52	31.40	195	7629844	22144	8.49	10.32	99	5179380	5542	1.33	1.56
8	345	50171476	1462503	1412.35	1932.42	347	14358756	101130	76.02	82.20	155	5539284	10453	3.28	3.72
12	more than 12.000 sec					499	28557588	239538	243.13	259.77	211	7308900	112408	34.03	35.52
16						651	38317620	737519	1429.94	1452.77	267	7494228	28188	12.61	14.59
32						1259	71885604	2591674	10584.69	10983.57	491	25946756	117920	116.23	124.47
64						more than 12.000 sec.					939	41330788	336080	774.05	799.94

Table 1: Pipelined ALU with 16 registers: Proving the validity of $\varphi_1 = AG\big(''\mathtt{R}_2{:=}\mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((\mathbf{R}_0\oplus\mathbf{R}_1) \equiv (AX)^3\mathbf{R}_2\big)\big)$ using Symbolic Z-Model Checking

	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	117	4939252	1372	0.52	0.64	122	5704084	1477	1.31	1.99	98	4966596	1140	0.70	0.78
4	193	5567460	13246	3.30	4.33	202	6742516	6480	5.53	11.32	154	5342868	8642	3.06	3.66
8	345	6631972	14748	10.93	13.21	362	17647956	5946	14.93	27.56	266	6758692	4136	5.20	6.30
12	497	37710484	197370	226.45	387.51	522	29638164	15642	50.33	88.70	378	12991844	7818	12.41	15.84
16	more than 12.000 sec.					682	32162244	16100	85.75	130.38	490	15205844	12197	18.82	24.53
32						1322	47493460	22383	586.19	775.10	938	41489060	15042	74.19	103.07
64						2602	45098852	61768	382.95	1823.86	1834	44027028	44606	278.53	391.74

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

ing realizability and for proving validity of incomplete designs (even if the Black Boxes lie inside the cone of influence for the considered CTL formula). 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.

At the moment we are working on further improvements concerning the accuracy of our approximate symbolic model checking. Starting from a concept for exact symbolic model checking of incomplete designs (containing several Black Boxes with bounded memory) we develop appropriate approximations trading off accuracy and computational resources.

7. 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, 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] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] T. Nopper and C. Scholl. Symbolic model checking for incomplete designs. Technical report, Albert-Ludwigs-University, Freiburg, May 2004.

[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.