Chapter 2
Verification of Incomplete Designs

Bernd Becker, Christoph Scholl and Ralf Wimmer

2.1 Introduction

With a hierarchical, component based design style unknown values increasingly emerge in different phases of the design and production process, and have to be handled by corresponding electronic design automation (EDA) tools. In the following we restrict our attention to the verification process and describe current state-of-the-art approaches that enable the handling of such unknown values using formal techniques: We provide a sequence of filters, trading off quality, and computation times of the analysis.

Unknown values in circuit verification occur, for instance, when a circuit is only partially available. Partially available means that for some of the circuit's components only their interface is known, i.e., the signals connected to the inputs and outputs of the components, but neither their internal structure nor the computed function. These missing parts are called *black boxes*. The actual values at their outputs are unknown. Verification has to take this into account.

There are different reasons for considering such incomplete circuits: Errors in a circuit design should be detected as early as possible; the later errors are corrected the higher are the incurred costs. Therefore it is desirable to apply verification techniques already in an early stage of the design process when not all parts of a circuit have been implemented yet.

B. Becker (✉) · C. Scholl · R. Wimmer
Institute of Computer Science, Albert-Ludwigs-Universität Freiburg,
Freiburg im Breisgau, Germany
e-mail: becker@informatik.uni-freiburg.de

C. Scholl
e-mail: scholl@informatik.uni-freiburg.de

R. Wimmer
e-mail: wimmer@informatik.uni-freiburg.de

© Springer International Publishing AG 2018

R. Drechsler (ed.), *Formal System Verification*, DOI 10.1007/978-3-319-57685-5_2

A further reason for considering incomplete circuits is that some modules like multipliers are notoriously hard to verify: If the property to be checked is expected to be independent of such a module, the module can be removed from the circuit, and instead it is checked whether the property under consideration holds for all possible replacements of the missing part. If this is the case, then the property also holds for the complete circuit. Otherwise either the remaining circuit is faulty or the removed module and the property interact in some unexpected way.

Considering incomplete circuits can also be beneficial for error diagnosis during debugging. Assume that an error is contained in one of the circuit's modules, but it is not known in which one. If, after removing one module, verification yields that there is an implementation of the removed part such that the considered property holds, then it is likely that the error is contained in the removed module.

If error diagnosis and error rectification are performed late in the design cycle when already a lot of efforts have been made to perform logic synthesis or even place and route steps for the complete design, then the question will be whether the design can be rectified by changing *locally* confined black boxes only, without introducing new connections to global signals leading to enormous costs for re-synthesis. A similar situation occurs in case of Engineering Change Order (ECO, small changes of specification late in the design cycle) where only locally confined parts (black boxes) should be replaced in order to satisfy the changed specification without sacrificing too much of the design efforts. In this case, it is particularly important to preserve the interface of the black boxes.

The synthesis of digital controllers [3, 4] that ensure certain properties of the system at hand can also be considered as a black-box verification problem: The controller to be synthesized is the black box, and one asks whether there is an implementation such that the given property holds.

Depending on the application there are two different problem classes that are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the complete design fulfills a certain property. On the other hand, *validity* asks whether the property holds no matter how the missing parts are implemented. Since validity and realizability are dual properties—a property φ is valid iff $\neg\varphi$ is not realizable—in the following we concentrate on realizability problems.

Our methods are based on checking the satisfiability of (quantified) Boolean formulas. In the meantime software tools checking the satisfiability of propositional Boolean formulas are well developed and have been proven to be very successful in many industrial applications [2]. In contrast to propositional formulae used for SAT, quantified Boolean formulae, where variables are existentially or universally quantified, allow a (more) succinct representation in many cases. The recent advances in the performance of QBF solvers, for example, conflict driven learning, resolution and expansion based algorithms [1], or preprocessing [16] enable more exact reasoning about realizability in presence of black boxes even for larger circuits. To be even more exact (or complete) in the general case dependencies between the quantified variables have to be taken into account. This leads to an extension of QBF, called *Dependency quantified Boolean formulas* (DQBF). DQBF allows existential

variables to depend on arbitrary sets of universal variables. DQBF solving currently is under development, according to the literature there exist currently two DQBF solvers: iDQ [12] and our tool HQS [15].

We now turn to the verification problems considered in this contribution and at first provide an overview of the problems and algorithms to be presented in more detail in the following sections.

The problem whether an incomplete combinational circuit can be completed such that it becomes equivalent to a given specification (*partial equivalence checking*, PEC) was first considered in [27] where several *approximate* and *exact* methods to solve the PEC problem have been presented. If an approximate algorithm reports that there is no implementation for the black boxes such that the specification holds, the desired specification is indeed not realizable. However, if such an algorithm is not able to prove non-realizability, this can be due to the approximate nature of the method, and the desired functionality may nevertheless be *not* realizable. The algorithms in [27] are based on solving SAT or QBF formulations of PEC. The SAT formulations are efficient to solve also due to a "near at hand" encoding, but also rather inaccurate due to a coarse approximation. Their accuracy is improved in several steps, leading to a QBF formulation that can solve PEC for a single black box exactly. In [27] additionally an exact characterization of realizability of PEC for multiple black boxes has been proposed (based on the decomposability of a certain Boolean relation). However, no feasible algorithmic method for solving the problem has been given.

Nevertheless, [27] was the first paper to consider an exact solution of the PEC problem taking into account that the *interfaces* of the black boxes in the incomplete circuit have to be preserved. Apart from the approach in [18, 19], the diagnosis and rectification problem respecting local interfaces has not been addressed in the literature so far. In [28, 29], e.g., rectifications are computed, but they are allowed to depend on arbitrary signals in the circuit. (Moreover, in contrast to [28, 29] uses a SAT formulation to compute rectifications for a given set of counterexamples only, without considering correctness for *all* possible inputs.) The approach of [18, 19] solves the PEC problem exactly, but it is restricted to problem instances of moderate sizes, since the black boxes are replaced by function tables using an exponential number of Boolean variables. A more efficient complete approach, is based on solving an extension of quantified Boolean formulas, so-called *dependency quantified Boolean formulas (DQBFs)* and was first presented in [14].

We have extended the application of realizability checking to sequential circuits which are specified by a set of properties (safety properties or more general properties formulated in Computation Tree Logic (CTL [9]). Here the question is whether an incomplete sequential design may be extended by black box implementations such that a set of given properties is satisfied. Also the problem of deciding validity is considered. We developed various approaches for solving the realizability problem either in an approximate or an exact manner. In the following, we discuss some representative approaches:

These approaches rely on a series of approximate methods with varying precision and costs for deciding the realizability of properties using symbolic methods. As in

the combinational case, the approximations are based on different methods to model the effect of the unknowns at the black box outputs to the overall circuit.

In [22] approximation methods are applied in the context of realizability checking of safety properties based on bounded model checking techniques (BMC—here a sequential circuit is "unrolled" for a number of time frames). This approximate approach leads to SAT or QBF problems. Here, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem. The approach is guided by proofs that non-realizability cannot be shown using the weaker methods, independently from the number of BMC unrollings, i.e., independently from the length of a counterexample which does not depend on the implementation of the black boxes. [23] extends [22] and provides proofs based on inductive arguments, showing that non-realizability can not be proven even by our most exact QBF based methods (also independently from the number of BMC unrollings). The approach of [23] provides an exact decision procedure for realizability in the case that the design contains exactly one black box which is allowed to read all input signals (which means that it has "complete information").

The usage of DQBF formulations for verifying incomplete designs was first proposed in [13, 14]. There we have shown that DQBF allows to check realizability precisely for combinational circuits with an arbitrary number of black boxes. This was generalized in to sequential circuits with combinational or bounded-memory black boxes.

In [24], we provided a series of approximate methods for deciding the realizability of CTL properties using symbolic methods. Moreover, [24] presents an exact method for deciding realizability for incomplete circuits with several black boxes under the assumption that the black boxes may contain only a bounded amount of memory. This exact method is based on introducing an exponential number of new variables and is therefore only suitable for small problem instances.

The remaining part of our contribution is structured as follows: In the following section, we introduce the necessary foundations that are required for verifiying incomplete circuits. In Sect. 2.3, we consider the case of incomplete combinational circuits, which do not contain any memory elements. The following Sect. 2.4 considers incomplete sequential circuits. It encompasses two parts: in Sect. 2.4.1 we consider techniques that use bounded model checking to refute realizability. Section 2.4.2 describes OBDD-based model checking algorithms for CTL properties. Finally, in Sect. 2.5 we conclude this paper.

2.2 Preliminaries

In this section, we provide a short overview of the underlying calculus used to solve the verification problems considered in the following. The interested reader is referred to [2] for more details.

Our algorithms are based on checking the satisfiability of (quantified) Boolean formulas. Even though deciding the satisfiability of a quantifier-free Boolean formula

(SAT) is an NP-complete problem [10], SAT solvers, i.e., software tools checking the satisfiability of quantifier-free Boolean formulas have been proven to be very successful in many industrial applications. The formula is typically provided in conjunctive normal form (CNF). A formula in CNF is a conjunction of clauses, and a clause is a disjunction of literals, where a literal is either a Boolean variable or its negation. For instance a clause is given by $(x \vee \neg y)$ with the Boolean variables x and y.

It will turn out that, depending on the problem considered, SAT may not allow an adequate concise description, and extensions like quantified Boolean formulas are more suitable:

Definition 2.1 (*Quantified Boolean formulas: Syntax*) Let V be a set of Boolean variables. A *quantified Boolean formula (QBF)* Ψ (in prenex form) has the form

$$\forall X_1 \exists Y_1 \forall X_2 \ldots \exists Y_k : \varphi,$$

where $X_1, Y_1, X_2, \ldots, Y_k$ are pairwise disjoint subsets of V such that $\bigcup_{i=1}^{k} X_i \cup Y_i = V$, $X_i \neq \varnothing$ for $i = 2, \ldots, k$, and $Y_j \neq \varnothing$ for $j = 1, \ldots, k-1$. φ is a Boolean formula over V, called the *matrix* of Ψ. $\forall X_1 \exists Y_1 \forall X_2 \ldots \exists Y_k$ is called Ψ's *quantifier prefix*.

Typically the matrix φ of a QBF is required to be in conjunctive normal form. A QBF has a linearly ordered quantifier prefix: each existential variable depends on all universal variables in whose scope it is. If we consider a QBF as a two-player game, then one player assigns the existential variables and the other player the universal ones. The game proceeds according to the quantifier prefix from left to right. The goal of the existential player is to satisfy the matrix, the universal player's goal to falsify it. Thereby, each player may base the assignment of the current variable on all assignments the opponent has made so far. The QBF is satisfied iff the existential player has a winning strategy which satisfies the matrix for all possible assignments of the universal variables.

Example 2.1 As an example consider the QBF $\forall x_1 \exists y_1 \forall x_2 \exists y_2 : \varphi$. It is satisfied if and only if: for *every* assignment of x_1, there *exists* an assignment for y_1 such that for *every* assignment of x_2 there *exists* an assignment for y_2, such that the matrix is satisfied. Obviously, here y_2 depends on x_2 and on x_1.

The complexity of QBF satisfiability is determined by the number of quantifier alternations between existential and universal quantifiers and vice versa in the prenex form. The general problem of QBF satisfiability is a PSPACE-complete problem [21].

In a next step, the dependencies between the quantified variables can be generalized: As already mentioned, in QBF each existential variable depends on all universal variables to the left. *Dependency quantified Boolean formulas* (DQBF) relax this restriction and allow existential variables to depend on arbitrary sets of universal variables.

Definition 2.2 (*Dependency quantified Boolean formulas: Syntax*) Let $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ be a set of Boolean variables and φ a quantifier-free Boolean formula over V. A *dependency quantified Boolean formula (DQBF)* ψ over V has the form

$$\psi = \forall x_1 \forall x_2 \ldots \forall x_n \exists y_y(D_{y_1}) \exists y_2(D_{y_2}) \ldots \exists y_m(D_{y_m}) : \varphi,$$

where $D_{y_i} \subseteq \{x_1, \ldots, x_n\}$ is the dependency set of y_i.

The semantics of such a formula is typically defined in terms of Skolem functions. For a set V of Boolean variables, let $\mathcal{A}(V) = \left\{\nu \,\middle|\, \nu : V \to \{0, 1\}\right\}$ denote the set of all variable assignments for V.

Definition 2.3 (*Dependency quantified Boolean formulas: Semantics*) Let ψ be a DQBF as defined above. It is *satisfied* if there are functions $s_{y_i} : \mathcal{A}(D_{y_i}) \to \mathbb{B}$ such that replacing all occurrences of y_i in φ by (a Boolean expression for) s_{y_i} for all $i = 1, \ldots, m$ turns φ into a tautology.

Example 2.2 Consider the following DQBF (with an appropriate matrix φ):

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi\,.$$

Here y_1 depends only on x_1 and y_2 only on x_2. There is no QBF prefix which expresses the same dependencies.

We will come back to this example in the following section where we will see that DQBF is an adequate formalism to model verification problems for incomplete combinational circuits.

In general, DQBF has a strictly higher expressiveness than QBF. This is also reflected in the complexity of the decision problem: deciding whether a DQBF is satisfiable is NEXPTIME-complete [25].

First DQBF solvers are currently under development: iDQ was the first available DQBF solver. It applies instantiation-based solving [12]. Our tool HQS [15] is currently the only other available DQBF solver. It uses quantifier elimination to turn the DQBF at hand into an equisatisfiable QBF, which can be solved by an arbitrary QBF solver. To reduce the computation times, HQS applies sophisticated preprocessing techniques. It is not only able to decide the satisfiability of DQBFs, but also to compute—in case the formula is satisfiable—Skolem functions for the existential variables [31].

2.3 Incomplete Combinational Circuits

In this chapter, we are going to investigate the verification of incomplete combinational circuits, i.e., circuits without memory elements containing unknown parts. After defining the partial equivalence checking problem, we investigate symbolic

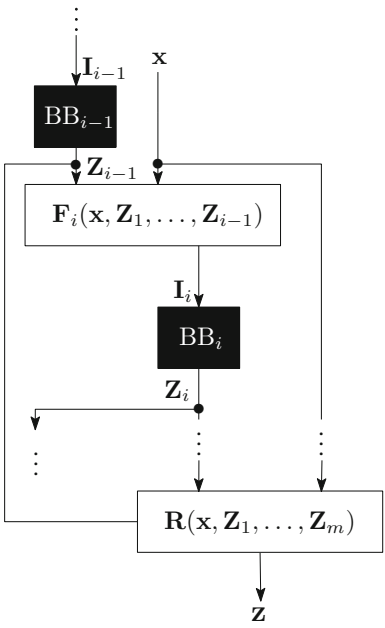

Fig. 2.1 Notation for incomplete combinational circuits

simulation with 01X-logic. We provide a sound formulation that enables an efficient, but incomplete solution method: If an answer is obtained, it is correct, but there are cases where the procedure cannot give a conclusive answer. We improve on this by using quantified Boolean formulas (QBF). For combinational circuits with only a single black box, the QBF formulation entails a complete decision procedure; for more than one black box, it is still an approximation. In this case, the extension of QBF to dependency quantified Boolean formulas (DQBF) has to be used.

2.3.1 The Partial Equivalence Checking Problem (PEC)

We first define incomplete circuits and the partial equivalence checking problem.

Figure 2.1 shows a part of an incomplete circuit. We assume that $\mathbf{x} = (x_1, \ldots, x_n)$ are the primary inputs of the circuit,[1] $\mathrm{BB}_1, \ldots, \mathrm{BB}_m$ are its unknown parts (so-called "black boxes"). In order to guarantee that the circuit is combinational for all possible combinational black box implementations, we assume that the black boxes are given in topological order, i.e., BB_i does not depend on BB_j for $1 \leq i < j \leq m$. Furthermore, we assume that the interfaces of the black boxes are known: $\mathbf{I}_i$ is the vector of input signals of BB_i and $\mathbf{Z}_i$ the vector of its output signals for $i = 1, \ldots, m$.

[1] In the following, we denote individual signals or variables in italic font (like x, y, z) and vectors in upright bold font (like $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$).

The known parts of the circuit are given by the vectors $\mathbf{F}_i(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1})$, which define $\mathbf{I}_i$, and the output functions $\mathbf{z} = \mathbf{R}(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)$ of the circuit.

Definition 2.4 (*Partial Equivalence Checking Problem*) The *Partial Equivalence Checking (PEC)* problem is defined as follows: Given an incomplete combinational circuit and a specification in form of a complete circuit, are there implementations of the black boxes such that both circuits become equivalent? In this case, the specification is said to be *realizable*.

Equivalent means that both circuits produce, for the same input pattern, the same output values. By a (negated) miter construction, we can combine the two circuits into a single one: corresponding inputs are driven by the same input signal, corresponding outputs are connected via equivalence gates (xnor-gates), and their outputs in turn via an and-gate. The single output of this miter circuit is 1 for an input pattern if both circuits compute the same value. The specification is unrealizable iff for all possible black box implementations there is an input pattern such that the output of the negated miter becomes 0.

In the following, we assume that the PEC is given as the negated miter circuit with the goal to make the output constantly 1 by implementing the black boxes appropriately.

2.3.2 SAT-based Approximations

One possibility to model the unknown behavior of the black boxes is to use a three-valued logic (also called 01X-logic), which adds a third value X to the classical Boolean logic. X stands for an unknown value. The Boolean operations can be generalized easily to the three-valued logic, e.g., by $0 \wedge \mathrm{X} = 0$, $1 \wedge X = \mathrm{X}$, etc. The complete truth tables for $\wedge$, $\vee$, and $\neg$ for the three-valued logic are shown in Table 2.1. Given a Boolean assignment for the primary inputs, we can assign X to the outputs of the black boxes and do a simulation of the circuit using the truth tables in Table 2.1. If the value of a circuit output is 0 or 1, this value is independent of the implementation of the black boxes. Obtaining output X means that, for the given input pattern, the output might depend on the black box implementation.

01X-logic is known to over-estimate the number of X-values in a circuit. Consider, for example, a circuit with one and- and one not-gate that computes $a \wedge (\neg a)$. The output of this circuit is 0 for all possible values of a, but in 01X-logic, if $a = \mathrm{X}$, the output is determined to be X.

Still, we can use 01X-logic to determine unrealizability of a PEC problem. To do so, we use symbolic simulation using the three-valued logic: Take the formula $z = R(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)$, which defines the output of the circuit, and check if, given an assignment of X to all black box outputs, there exists an assignment of $\mathbf{x}$ (in $\{0, 1\}$) such that z has the value 0 (or equivalently $\neg z$ value 1):

Table 2.1 Boolean operations on 01X-logic

(a) Conjunction			
$\wedge$	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X
(b) Disjunction			
$\vee$	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X
(c) Negation			
$\neg$	0	1	X
	1	0	X

$$\big(z \equiv R(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)\big) \wedge \big(x_1 \neq \mathrm{X}\big) \wedge \cdots \wedge \big(x_n \neq \mathrm{X}\big) \wedge \big(\mathbf{Z}_1 = \mathbf{X}\big) \wedge \cdots \wedge \big(\mathbf{Z}_m = \mathbf{X}\big) \wedge \big(z = 0\big).$$

This is a satisfiability (SAT) problem over 01X-logic. To make standard SAT solvers applicable, we have to re-encode the formula over standard Boolean logic. Jain et al. [17] propose to use $\lceil \log_2 3 \rceil = 2$ Boolean variables to encode the three different values 0, 1, and X: 0 is encoded as (1, 0), 1 as (0, 1), and X as (0, 0). For three-valued variables a, b with encodings (a^0, a^1) and (b^0, b^1), resp., we obtain: $a \vee b = (a^0 \wedge b^0, a^1 \vee b^1)$, $a \wedge b = (a^0 \vee b^0, a^1 \wedge b^1)$, and $\neg a = (a^1, a^0)$. Of course, this encoding has to be applied only to those signals which are in the cone of influence of the black box outputs. As a Boolean signal a is equivalent to the encoding $(\neg a, a)$, we can convert standard Boolean signals into encoded three-valued ones where necessary.

The result of encoding the three-valued miter circuit is a Boolean circuit with two outputs (z^0, z^1) with $z^0 \equiv R^0(\mathbf{x}^0, \mathbf{x}^1, \mathbf{Z}_1^0, \mathbf{Z}_1^1, \ldots, \mathbf{Z}_m^0, \mathbf{Z}_m^1)$ and $z^1 \equiv R^1(\mathbf{x}^0, \mathbf{x}^1, \mathbf{Z}_1^0, \mathbf{Z}_1^1, \ldots, \mathbf{Z}_m^0, \mathbf{Z}_m^1)$. By adding appropriate clauses to the SAT formula, we have to enforce that the following restrictions are fulfilled: (1) $z^0 \wedge \neg z^1$, i.e., $(z^0, z^1) = (1, 0)$, (2) $\neg Z_{i,j}^0 \wedge \neg Z_{i,j}^1$ for $i = 1, \ldots, m$ and $Z_{i,j} \in \mathbf{Z}_i$, i.e., all black box outputs are assigned to X, and (3) $x_i^0 \oplus x_i^1 = 1$ for $i = 1, \ldots, n$, i.e., $(x_i^0, x_i^1) \in \{(0, 1), (1, 0)\}$.

If the resulting Boolean formula is satisfiable, there is an input pattern for the circuit (which corresponds to the assignment of $(x_1^1, \ldots, x_n^1)$) that leads to an output of $0 = (1, 0)$, independent of the actual implementation of the black boxes. Hence, the PEC is unrealizable. Contrary, if the formula is unsatisfiable, we cannot conclude realizability: We cannot distinguish whether the design is realizable or the applied method too weak, caused by one of the following limitations:

- The 01X-logic is not precise in case of reconvergences of the circuit.
- Using 01X-logic, we search for counterexample patterns (refuting realizability) that are independent of the black boxes. However, in general, we might need a different pattern for each implementation of the black boxes.
- We ignore the input cones of the black boxes.

This is improved step by step in the following section by using quantified Boolean formulas.

2.3.3 QBF-based Methods

The problem that 01X-logic is inaccurate in case of reconvergences can be solved by using quantified Boolean formulas. The idea is to check if for all assignments of the primary inputs there are values of the black box outputs such that the formula $R(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)$ evaluates to 1. This yields the following QBF:

$$\forall \mathbf{x} \exists \mathbf{Z}_1 \ldots \exists \mathbf{Z}_m : \big(z \equiv R(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)\big) \wedge \big(z = 1\big). \tag{2.1}$$

Unsatisfiability of this formula implies unrealizability of the PEC problem. This formulation is more precise than the 01X-based formulation, but typically more expensive to solve—while SAT is NP-complete [10], QBF is PSPACE-complete [21]. This also affects the solver performance from a practical point of view: SAT problems that can be solved nowadays may be roughly two orders of magnitude larger than solvable QBFs.

In spite of the higher precision, (2.1) still does not constitute a complete decision method for PEC: so far, we have neglected the input cones of the black boxes.

If the black boxes are not directly connected to the primary inputs but to internal signals we have to take into account that not all possible combinations of values may arrive at the inputs of the black boxes, i.e., that the black boxes have only partial information about the primary inputs.

Since we use universal quantification for the black box inputs, we only have to ensure that our formula is satisfied, if the value of the black box inputs $\mathbf{I}_i$ does not deviate from the values obtained as a function $\mathbf{I}_i = \mathbf{F}_i(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1})$. The following matrix fulfills this requirement:

$$\varphi = \big(\mathbf{I}_1 \not\equiv \mathbf{F}_1(\mathbf{x})\big) \vee \cdots \vee \big(\mathbf{I}_m \not\equiv \mathbf{F}_m(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_{m-1})\big) \vee R(\mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)\,. \tag{2.2}$$

We use universal quantification for the primary inputs $\mathbf{x}$ and all black box inputs $\mathbf{I}_1, \ldots, \mathbf{I}_m$. The outputs $\mathbf{Z}_1, \ldots, \mathbf{Z}_m$ of the black boxes are existentially quantified. If a black box output is the input of another black box, we introduce a buffer, "computing" the identity function, to avoid conflicts. We have to take care that in the quantifier

prefix, all inputs of a black box precede its outputs. Then there are still several prefixes possible, e.g.,

$$\forall \mathbf{x} \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_m \exists \mathbf{Z}_1 \ldots \exists \mathbf{Z}_m : \varphi, \tag{2.3}$$

$$\forall \mathbf{I}_1 \exists \mathbf{Z}_1 \forall \mathbf{I}_2 \backslash \mathbf{I}_1 \exists \mathbf{Z}_2 \ldots \exists \mathbf{Z}_m . \forall \mathbf{x} \backslash (\mathbf{I}_1 \cup \mathbf{I}_2 \cup \cdots \cup \mathbf{I}_m) : \varphi \tag{2.4}$$

The QBF with prefix (2.3) contains only two quantifier blocks. It is therefore typically easier to solve than (2.4). The drawback is that is less precise: In a QBF, an existential variable depends on all universal ones left of it in the prefix. In (2.4), each existential variable in general depends on fewer variables than in (2.3). Additionally, (2.4) is precise in case of a single black box. For more than one black box, (2.4) is not able to express the dependencies of the black box outputs on their corresponding inputs exactly.

2.3.4 DQBF-based Methods

In the previous two sections, we made the decision procedure for PEC more and more precise. One last step is missing in order to obtain a complete and sound decision procedure for PEC.

If a combinational design contains more than one black box and if the black boxes do not have the same interfaces, it is typically not possible to express the dependencies of the black boxes' outputs on their inputs in QBF exactly. This is illustrated in the following example.

Example 2.3 Consider an incomplete design with two black boxes BB_1 and BB_2. The output Z_1 of black box BB_1 depends only on x_1, the output Z_2 of BB_2 only on x_2.[2] Then three different QBF formulation are possible (with an appropriate matrix φ): (1) $\forall x_1 \exists Z_1 \forall x_2 \exists Z_2 : \varphi$, (2) $\forall x_2 \exists Z_2 \forall x_1 \exists Z_1 : \varphi$, and (3) $\forall x_1 \forall x_2 \exists Z_1 \exists Z_2 : \varphi$, In formula (1), Z_2 depends not only on x_2, but also on x_1, violating the interface specification of BB_2. The same holds in (2) for BB_1, which would be allowed to read x_2. Finally, in (3) both black boxes read both signals.

To be able to express the dependencies of the black box outputs exactly, we have to resort to *dependency quantified Boolean formulas (DQBFs)*, as already introduced in the previous section. While a QBF has a linearly ordered quantifier prefix—each existential variable depends on all universal variables in whose scope it is—in a DQBF the existential variables are explicitly annotated with the universal variables they depend upon.

[2] In this example we assume that the inputs of the black boxes are directly connected to the primary inputs. Thus we do not need to introduce separate vectors $\mathbf{I}_1$ and $\mathbf{I}_2$ for the black box inputs.

Example 2.4 Consider again the incomplete design from Example 2.3. It leads to the following DQBF (with an appropriate matrix φ):

$$\forall x_1 \forall x_2 \exists Z_1(x_1) \exists Z_2(x_2) : \varphi\,.$$

Here Z_1 depends on x_1 and Z_2 on x_2, which coincides with the interface specification of the incomplete design.

In general, we can use the following DQBF formulation for PEC:

$$\psi = \forall \mathbf{x} \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_m \exists \mathbf{Z}_1(\mathbf{I}_1) \ldots \exists \mathbf{Z}_m(\mathbf{I}_m) : \varphi, \tag{2.5}$$

where the matrix φ is exactly the same as in Eq. (2.2) defined for QBF. In contrast to QBF, DQBF allows us to model arbitrary black box interfaces precisely.

It is not hard to show that there is not only a linear transformation from PEC to DQBF, but also vice versa [13, 14]. The existence of these transformations implies on the one hand that PEC and DQBF have the same complexity. They are both NEXPTIME-complete. On the other hand, we can solve PEC exactly by solving an appropriate DQBF formulation.

2.4 Incomplete Sequential Circuits

We investigate how and to which extent the techniques presented in the previous section for incomplete combinational circuits can be generalized to incomplete circuits, which contain memory elements, i.e., incomplete sequential circuits.

In contrast to combinational circuits, we consider the validity and realizability of a property regarding an incomplete sequential circuit and then present approaches for the analysis of those circuits.

Analogously to the combinational case an *incomplete sequential circuit* is a sequential circuit containing black boxes. See Fig. 2.2 for an illustration. The circuit contains m black boxes $\mathrm{BB}_1, \ldots, \mathrm{BB}_m$, shown as black rectangles. Their input signals are denoted by $\mathbf{I}_1, \ldots, \mathbf{I}_m$ and their output signals by $\mathbf{Z}_1, \ldots, \mathbf{Z}_m$. The primary inputs of the circuit are $\mathbf{x} = (x_0, \ldots, x_n)$, the current state is given by the signals $\mathbf{s} = (s_0, \ldots, s_r)$. The input cones of the black boxes compute the functions $\mathbf{I}_i = \mathbf{F}_i(\mathbf{x}, \mathbf{s}, \mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1})$. We thereby assume that there are no cyclic dependencies between the black boxes and that they are topologically ordered, i.e., BB_i only depends on the values computed by $\mathrm{BB}_1, \ldots, \mathrm{BB}_{i-1}$. To simplify notation, we assume w. l. o. g. that no black box output is directly connected to a black box input, i.e., $\mathbf{Z}_i$ is disjoint from $\mathbf{I}_j$ for all i, j. If this is not the case, we insert a buffer between the corresponding black boxes, which does not modify the functionality of the circuit. Finally, the output $\mathbf{y}$ and the next state $\mathbf{s}'$ of the circuit are given by the Boolean functions $(\mathbf{y}, \mathbf{s}') = \mathbf{R}(\mathbf{x}, \mathbf{s}, \mathbf{Z}_1, \ldots, \mathbf{Z}_m)$.

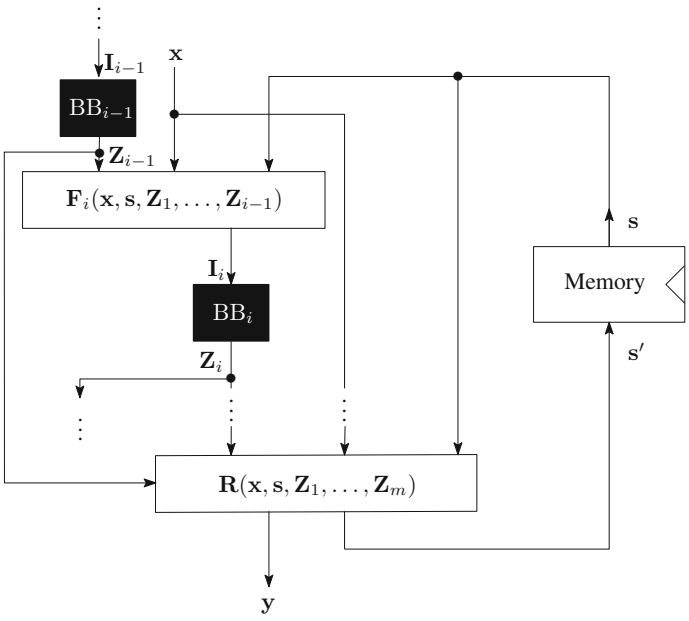

Fig. 2.2 Notations for an incomplete sequential circuit

The complexity of deciding realizability depends on the allowed behavior of the black boxes: If we assume that the contents of the black boxes are combinational circuits, the problem of deciding realizability is NEXPTIME-complete. If we allow the black boxes to contain an arbitrary amount of memory, the interesting decision problems (see below) become undecidable [26]. The case of black boxes with a bounded amount of memory (i.e., we know an upper bound on the number of bits the black boxes can store internally) can be reduced to the case of combinational black boxes by adding the memory of the black boxes to the surrounding circuit such that these memory cells are read and written only by the corresponding black box.

For a given property P two questions regarding a partial circuit are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the complete circuit satisfies P. On the other hand, *validity* asks whether P is satisfied for all possible implementations. Since validity of P is given iff $\neg P$ is not realizable, we restrict ourselves in the following to realizability problems.

In particular, in the following subsection we present approaches based on bounded model checking techniques (BMC) [22, 23]. As properties we consider invariant properties: Given a Boolean formula $P(\mathbf{x}, \mathbf{s}, \mathbf{y})$, which describes the states of the circuit that satisfy the invariant, are there implementations of the black boxes such that $P(\mathbf{x}, \mathbf{s}, \mathbf{y})$ is satisfied in each step of the circuit? For more general classes of properties like arbitrary CTL properties, we refer the reader to Subsect. 2.4.2.

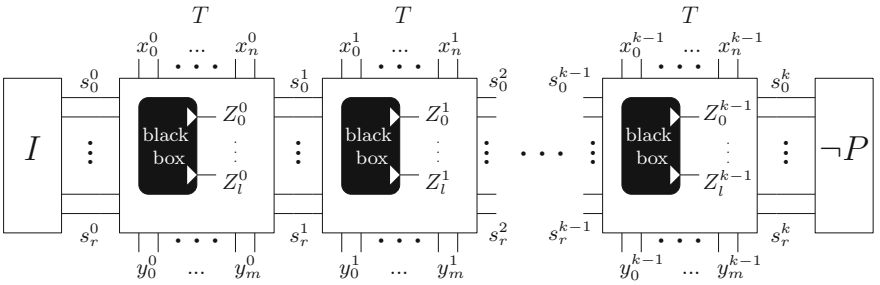

Fig. 2.3 Encoding of the BMC problem for incomplete designs [23]

2.4.1 BMC for Incomplete Designs

BMC for incomplete designs aims to refute the realizability of a property, that is, it tells the designer, no matter how the unknown parts of the system will be implemented, the property will always fail. To put it in other words, the error is already in the implemented system. If this is the case, then we call the property P *unrealizable*. Here we restrict the properties to *invariants*. In a first formulation, we make use of QBF modeling where the variables representing the black box outputs are universally quantified. We even allow black box replacements to produce different output values for the same input values at different time steps which simplify the formula but is a source of inexactness in case that only combinational circuits are allowed as black box implementations and also in case of black boxes with a bounded amount of memory[3]—and thus the method might miss to prove the unrealizability of some properties. Moreover, we first confine ourselves to single black boxes. If several black boxes with dedicated interfaces are combined into a single black box, then this is another source of inexactness. To encode the BMC problem of incomplete designs, we are naming the variables as shown in Fig. 2.3. We use an upper index to specify the time instance of a variable. s_i^j denotes the i-th state bit in the j-th unfolding (let $\mathbf{s}^j = s_0^j, \ldots, s_r^j$). The same holds for the primary inputs $\mathbf{x}^j = x_0^j, \ldots, x_n^j$, the primary outputs $\mathbf{y}^j = y_0^j, \ldots, y_m^j$, and the black box outputs $\mathbf{Z}^j = Z_0^j, \ldots, Z_l^j$. The next state variables $\mathbf{s}^{j+1}$ depend on the current state, the primary inputs and the black box outputs. The whole circuit is transformed according to [30] using additional auxiliary variables $\mathbf{H}^j$ for each unfolding depth j. The predicate describing the initial states is given by $I(\mathbf{s}^0)$. Since we assume a single initial state in this paper, the initial state $I(\mathbf{s}^0)$ is encoded by unit clauses, setting the respective state bits to their initial value. The transition relation of time frame i is given by $T(\mathbf{s}^{i-1}, \mathbf{x}^{i-1}, \mathbf{Z}^{i-1}, \mathbf{s}^i)$. The invariant $P(\mathbf{s}^k)$ is a Boolean expression over the state variables[4] of the k-th unfolding.

[3] Black boxes with a bounded amount of memory are reduced to the case of combinational black boxes by adding the memory of the black boxes to the surrounding circuit as mentioned above.

[4] In general the property can also check the primary inputs and outputs, but for sake of convenience, we omit details here.

Using this information, the quantifier prefix (and the matrix) for the unrealizability problem results in the QBF formula (2.6). For the sake of simplicity we include the variables representing the primary outputs of unfolding depth j into $\mathbf{H}^j$.

$$\begin{array}{lllll} BMC(k) := & \exists\, \mathbf{s}^0\, \mathbf{x}^0 & \forall\, \mathbf{Z}^0 & \exists\, \mathbf{H}^0 \\ & \exists\, \mathbf{s}^1\, \mathbf{x}^1 & \forall\, \mathbf{Z}^1 & \exists\, \mathbf{H}^1 \\ & & \vdots & \\ & \exists\, \mathbf{s}^{k-1}\, \mathbf{x}^{k-1} & \forall\, \mathbf{Z}^{k-1} & \exists\, \mathbf{H}^{k-1} \\ & \exists\, \mathbf{s}^k & & \end{array}$$

$$I(\mathbf{s}^0) \wedge \bigwedge_{i=1}^{k} T(\mathbf{s}^{i-1}, \mathbf{x}^{i-1}, \mathbf{Z}^{i-1}, \mathbf{s}^i) \wedge \neg P(\mathbf{s}^k) \tag{2.6}$$

The semantics following from the prefix corresponds to the following question:

> Does there exist a state $\mathbf{s}^0 = s_0^0, \ldots, s_r^0$ and an input vector $\mathbf{x}^0 = x_0^0, \ldots, x_n^0$ at depth 0 such that for all possible values of the black box outputs $\mathbf{Z}^0 = Z_0^0, \ldots, Z_m^0$ there exists an assignment to all auxiliary variables $\mathbf{H}^0$ (resulting in a next state $\mathbf{s}^1 = s_0^1, \ldots, s_r^1$) and an input vector $\mathbf{x}^1 = x_0^1, \ldots, x_n^1$ at depth 1, etc. such that the property is violated at time frame k?

The BMC procedure iteratively unfolds the incomplete circuit for $k = 0, \ldots, K$ until a predefined maximal unfolding depth K is reached. If a QBF solver finds $BMC(k)$ satisfiable, the unrealizability of the property P has been proven. In that case, the resulting system can reach a "bad state" after k steps, no matter how the black box is implemented.

We can prove that, whenever $BMC(k)$ is *unsatisfiable*, there is an implementation of the black box (possibly with a limited number of memory elements) which is able to avoid error paths of length k *as long as the black box is allowed to read all primary inputs*. However, if the black box in the design at hand is not directly connected to all primary inputs, (i.e., if the black box does not have "complete information"), such an implementation does not need to exist. Thus, for black boxes having "incomplete information" the property may be unrealizable although BMC with QBF modeling is not able to prove this. In this case, DQBF is necessary to express the actual dependencies of the black boxes on their inputs.

In the following, we give an example illustrating the approach:

Example 2.5 Consider the incomplete circuit shown in Fig. 2.4. The state bits s_0 and s_1 depend on the current state, the primary input x, and the black box outputs Z_0 and Z_1, respectively, and are computed by the transition functions $s_0' = x \vee Z_0$ and $s_1' = (x \wedge Z_1) \vee (s_0 \wedge \neg Z_1)$. Let the invariant property $P = \neg(s_0 \wedge s_1)$ state that s_0 and s_1 must never be 1 at the same time. Let the initial state of the system be defined as $s_0^0 = s_1^0 = 0$. After checking for an initial violation of the property, the BMC procedure unfolds the system once, and tries to find an assignment to x^0 such that for

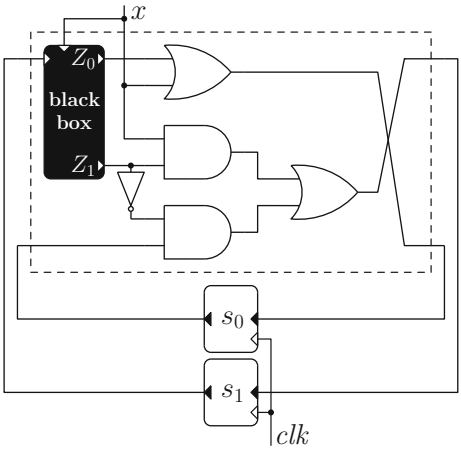

Fig. 2.4 Example of an incomplete design [23]

all possible assignments to Z_0^0 and Z_1^0 the state (1, 1) can be reached. Indeed, $x^0 = 1$ implies $s_0 = 1$ for all assignments to the black box outputs, however, for $Z_1^0 = 0$ $s_1 = 1$ cannot be obtained (neither by setting $x^0 = 0$ nor $x^0 = 1$). Thus, $BMC(1)$ is unsatisfiable and BMC continues by adding a second copy of the transition relation to the problem. If $x^0 = 1$, the current state bit s_0^1 at the second unfolding evaluates to 1 as well. Furthermore, if x^1 is set to 1, the next state bits s_0^2 and s_1^2 evaluate to 1 for all values of Z_0^1 and Z_1^1. Hence, when applying the input pattern $x^0 = x^1 = 1$, a state violating P can be reached after two steps for all actions of the black box and thus, $BMC(2)$ is satisfiable and P is unrealizable.

2.4.1.1 SAT-based Approximations and 01X-Hardness

As already shown in previous sections, QBF can be approximated using 01X-logic. In the previous example, it was not necessary to use QBF encoding for Z_0. When applying $Z_0 = X$, unrealizability still can be proven by applying $x^0 = x^1 = 1$.

Since the problem instances using 01X-modeling are typically easier to solve, we are using the following verification flow:

Given an incomplete design and an invariant, we start the BMC process with a pure 01X-modeling, that is we extend Boolean logic by a third value 'X' which then is applied to all black box outputs. Using the two-valued encoding proposed by Jain (cf. Sect. 2.3.2), the BMC unfoldings still yield SAT problems which can be solved by a state-of-the-art SAT solver. However, 01X-modeling may be too coarse to prove unrealizability leading to unsatisfiable BMC instances for every unfolding depth (we call such verification problems *01X-hard*). In [22] we presented a method based on Craig interpolation to classify 01X-hard problems on-the-fly along the BMC process, thus preventing the solver running into unsatisfiable instances forever. Additionally, the computed Craig interpolants provide information about

the origin of the 01X-hardness, and a subset of the black box outputs which have to be modeled more precisely using QBF is heuristically determined. Now a QBF-based BMC tool processes the information gathered from the Craig interpolants and uses one universally quantified variable for each black box output which needs a more precise modeling. Using a combined 01X/QBF-modeling (or a pure QBF-modeling) the BMC unfoldings yield QBF formulas. In that way, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem.

2.4.1.2 QBF-based Approximations and QBF-hardness

However, even when using the more precise QBF modeling technique to model the unknown behavior of the black box, no conclusive result is guaranteed. At this point, an extension given in [23] is introduced into the workflow. Similar to 01X-hardness for 01X-modeled incomplete designs, a QBF modeled BMC problem can now be classified as *QBF-hard*, if QBF-based BMC would continuously run into unsatisfiable unfoldings.

As already discussed above, under certain conditions (black boxes having "incomplete information") the BMC procedure using a QBF formulation is not able to prove unrealizability even if the property is indeed unrealizable. In this sense, the QBF formulation is a sound but incomplete approximation (just as 01X-modeling which is also an approximation, but is strictly coarser). If unrealizability cannot be proven due to the approximative nature of the method or if the property is really realizable, then the BMC procedure described above would produce unsatisfiable QBF formulas for all unfoldings and would never return a result.

The idea of proving QBF-hardness is as follows: The QBF-based BMC procedure classifies a property as unrealizable iff there exist input sequences of some length k such that independently from the black box actions the property will be violated after k steps. Conversely, the QBF-based BMC procedure is not able to prove unrealizability with an unfolding of length k or smaller, if for each input valuation in each time frame there is an action of the black box such that the property is fulfilled after k steps, and additionally all states on these paths also fulfill the property. Furthermore, if we can prove for this scenario that after at most k steps every state has already been visited before, we can be sure that the QBF-based BMC procedure will *never* produce a satisfiable instance, since for every input pattern it is possible to determine at least one realization of the black box leading to a state which does not violate the property, independently from the length of the unfolding.

This concept is illustrated in Fig. 2.5. Let $\mathbf{s}^0 = \boldsymbol{\alpha}_1^0$ be the initial state which fulfills P. Next, the graph branches for all possible assignments $\boldsymbol{\iota}_1^0, \ldots, \boldsymbol{\iota}_m^0$ to the primary inputs $\mathbf{x}^0$. For each of these values $\boldsymbol{\iota}_i^0$ there exists an action of the black box outputs $\mathbf{Z}^0 = \mathbf{l}_i^0$ leading to next states $\mathbf{s}^1 = \boldsymbol{\alpha}_i^1$ which all fulfill P. Once a state is equivalent to a state which was visited before (which is indicated by a dashed backward arrow in Fig. 2.5 stating that $\boldsymbol{\alpha}_1^1 = \boldsymbol{\alpha}_1^0$, $\boldsymbol{\alpha}_1^2 = \boldsymbol{\alpha}_2^1$, $\boldsymbol{\alpha}_m^2 = \boldsymbol{\alpha}_1^0$, respectively), this branch does not need to be further explored. If at some depth all so far explored

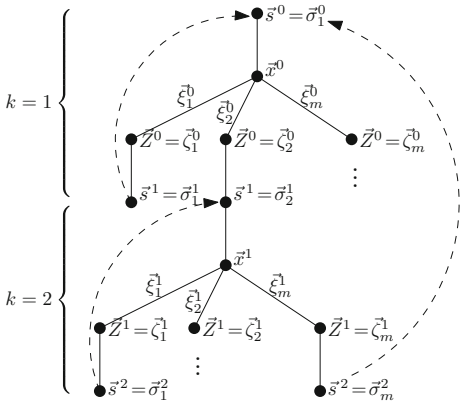

Fig. 2.5 QBF-hardness graph [23]

states point back to already visited states, then the black box outputs are set in a way that the system remains in "good states" forever, i.e., we are in the situation sketched above and we can be sure that the QBF-based BMC procedure will never produce a satisfiable instance, independently from the length of the unfolding. Thus, determining whether a graph fulfilling the aforementioned properties exists answers the question of whether a design is QBF-hard.

In [23] it has been shown that the existence of a QBF-hardness graph can be checked using a series of QBF formulas. Once the QBF-hardness of the design under verification is proven, two options are considered. In case of a combined 01X/QBF-modeling an abstraction refinement procedure will identify more black box outputs for QBF-modeling (in an extreme case yielding a pure QBF-modeled problem) and repeat the QBF-based BMC procedure. If all black box outputs are already QBF-modeled, one has to resort to DQBF modeling.

2.4.1.3 DQBF-based Modeling

If no upper bound on the amount of memory in the black boxes is known, one has to use BMC with DQBF modeling. The dependency sets of the black box outputs are chosen in the following way: If black box BB_i with outputs $\mathbf{Z}_i$ reads the signals $\mathbf{I}_i$, then in the formula $BMC(k)$, the dependency set $D_{\mathbf{Z}_i^j}$ of $\mathbf{Z}_i$ in the j-th unrolling contains all instances of $\mathbf{I}_i$ up to j, i.e., $D_{\mathbf{Z}_i^j} = \{\mathbf{I}_i^k \mid k = 0, \ldots, j\}$. This is exactly the information that has been read by BB_i up to the j-th unrolling. In the worst case, all of this information has been stored in the black boxes' internal memory and the output in the j-th step may depend on it.

Still, this approach is an incomplete decision procedure as the decision problem itself is in general undecidable.

However, if we assume that the black box implementations are combinational circuits, i.e., that the black boxes do not contain any memory elements, realizability

can be formulated as a DQBF without the necessity to unroll the circuit. The following formula is satisfiable iff the sequential circuit is realizable with combinational implementations of the black boxes:

$$\begin{aligned}&\forall \mathbf{s}\, \forall \mathbf{s}' \forall \mathbf{x}\, \forall \mathbf{I}_1 \ldots \forall \mathbf{I}_n\, \exists \mathbf{Z}_1(\mathbf{I}_1) \ldots \exists \mathbf{Z}_n(\mathbf{I}_n)\, \exists w(\mathbf{s})\, \exists w'(\mathbf{s}') : \\ &\qquad \big(I(\mathbf{s}) \Rightarrow w\big) \wedge \big(w \Rightarrow P(\mathbf{s})\big) \wedge \big(\mathbf{s} \equiv \mathbf{s}' \Rightarrow w \equiv w'\big) \wedge \\ &\Big(\big(w \wedge \bigwedge_{i=1}^{n} (\mathbf{I}_i \equiv \mathbf{F}_i(\mathbf{x}, \mathbf{s}, \mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1})) \wedge T(\mathbf{s}, \mathbf{x}, \mathbf{Z}_1, \ldots, \mathbf{Z}_n, \mathbf{s}')\big) \Rightarrow w'\Big).\end{aligned} \tag{2.7}$$

This formula is based on the notion of a winning set: A subset $W \subseteq S$ of the states of the considered circuit is a *winning set* if all states in W satisfy the invariant P and, for all values of the primary inputs, the black boxes can ensure (by computing appropriate values) that the successor state is again in W.

A given incomplete sequential circuit is realizable if there is a winning set that includes the initial state of the circuit. This can be formulated as the DQBF (2.7). Similar to the combinational case, we have to take into account that the black boxes are typically not directly connected to the primary inputs, but to internal signals. This is done by restricting the requirement that the successor state is again a winning state to the case when the black box inputs are assigned consistently with the values computed by their input cones.

This formula (together with the corresponding result for incomplete combinational circuits, see Sect. 2.3.4) also shows that deciding the realizability of invariants for

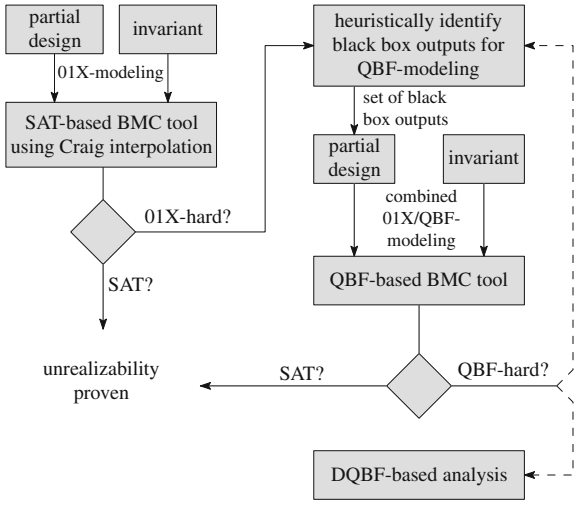

Fig. 2.6 Workflow

incomplete sequential circuits with combinational (or bounded-memory) black boxes is NEXPTIME-complete.

Figure 2.6 illustrates the complete verification flow for incomplete sequential circuits.

2.4.2 Model Checking for Incomplete Designs

In this section, we consider methods for incomplete designs which do not consider *bounded* model checking as in Sect. 2.4.1, but *symbolic* model checking using symbolic state set representations, i.e., state set representations based on binary decision diagrams (BDDs) [6] or and-inverter-graphs (AIGs). The approach generalizes symbolic model checking for complete designs [8] to (approximate) symbolic model checking for *incomplete* designs. In doing so, we can extend the consideration of invariant properties to the consideration of general CTL (computation tree logic) properties [9].

2.4.2.1 Symbolic Model Checking for Complete Designs

Symbolic model checking for complete designs [8] is applied to Kripke structures on the one hand, which may be derived from sequential circuits, and to a formula of a temporal logic (in our case: computation tree logic, CTL) on the other hand. A (complete) sequential circuit (such as considered in the section before) defines a Mealy automaton $M = (\mathbb{B}^{|\mathbf{s}|}, \mathbb{B}^{|\mathbf{x}|}, \mathbb{B}^{|\mathbf{y}|}, \delta, \lambda, \mathbf{s}^0)$ with state set $\mathbb{B}^{|\mathbf{s}|}$, the set of inputs $\mathbb{B}^{|\mathbf{x}|}$, the set of outputs $\mathbb{B}^{|\mathbf{y}|}$, transition function $\delta : \mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|} \to \mathbb{B}^{|\mathbf{s}|}$, output function $\lambda : \mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|} \to \mathbb{B}^{|\mathbf{y}|}$ and initial state $\mathbf{s}^0 \in \mathbb{B}^{|\mathbf{s}|}$.[5] 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 $struct(M) = (S, R, L)$ where $S = \mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|}$, $R \subseteq S \times S$, $L : S \to V$. V is the set of atomic properties, i.e., $V = \{x_0, \ldots, x_{|\mathbf{x}|-1}\} \cup \{y_0, \ldots, y_{|\mathbf{y}|-1}\}$, $R = \left\{\left((\mathbf{s}, \mathbf{x}), (\mathbf{s}', \mathbf{x}')\right) \,\middle|\, \mathbf{s}, \mathbf{s}' \in \mathbb{B}^{|\mathbf{s}|}, \mathbf{x}, \mathbf{x}' \in \mathbb{B}^{|\mathbf{x}|}, \delta(\mathbf{s}, \mathbf{x}) = \mathbf{s}'\right\}$, and $L\big((\mathbf{s}, \mathbf{''})\big) = \left\{x_i \,\middle|\, \varepsilon_i = 1\right\} \cup \left\{y_i \,\middle|\, \lambda_i(\mathbf{s}, \mathbf{''}) = 1\right\}$.

Given a set V of atomic propositions, the syntax of CTL formulas is given by the following context-free grammar, where $v \in V$ is an atomic proposition and Φ the only non-termination symbol:

$$\Phi ::= v \mid \neg\Phi \mid (\Phi \vee \Phi) \mid EX\Phi \mid EG\Phi \mid E\Phi U\Phi.$$

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

[5] Here, we assume to have exactly one initial state; it is trivial to extend the following to sets of initial states.

(a) Fixed point iteration for EG

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

(b) Fixed point iteration for EU

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

Fig. 2.7 Fixed point iteration algorithms

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

Further CTL operations like $\wedge$, EF, AX, AU, AG, and AF can be expressed by using $\neg$, $\vee$, EX, EU, and EG [20].

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 $\chi_R(\mathbf{s}, \mathbf{x}, \mathbf{s}') := \prod_{i=0}^{|\mathbf{s}|-1} \left(\delta_i(\mathbf{s}, \mathbf{x}) \equiv s_i'\right)$ of the transition relation R:

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

with

$$
\chi_{EX}(\chi_X)(\mathbf{s}, \mathbf{x}) := \exists \mathbf{s}' \exists \mathbf{x}' \left(\chi_R(\mathbf{s}, \mathbf{x}, \mathbf{s}') \cdot \left(\chi_X |_{\substack{\mathbf{s} \leftarrow \mathbf{s}' \\ \mathbf{x} \leftarrow \mathbf{x}'}}\right)(\mathbf{s}', \mathbf{x}') \right)
$$

χ_{EG} and χ_{EU} can be evaluated by the fixed point iteration algorithms shown in Fig. 2.7.

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 $\mathbf{s}^0$ of M:

$$M \models \varphi :\Longleftrightarrow \forall \mathbf{x} \in \mathbb{B}^{|\mathbf{x}|} : struct(M), (\mathbf{s}^0, \mathbf{x}) \models \varphi \iff \left(\forall \mathbf{x}(\chi_{Sat(\varphi)}|_{\mathbf{s}=\mathbf{s}^0})\right) = 1.$$

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

(1) Flexible modeling of black box outputs in symbolic simulation
For symbolic CTL model checking of a given design, a symbolic representation of its output function λ and of its transition function δ are needed first. In order to generalize CTL model checking to *incomplete* designs, the potential effect of the black box outputs to the remaining design needs to be modeled in order to compute λ and δ. As in the sections before, we consider two different methods modeling black box outputs with differing accuracy: The first one (symbolic $(0, 1, X)$-simulation) is based on ternary $(0, 1, X)$ logic and the second one (symbolic Z_i-simulation) introduces for each output of a black box a separate variable Z_i. Similarly to Sect. 2.4.1.1, we consider a method for flexible modeling of different black box outputs by differing methods. This method will be applied later on for our approximate model checking algorithm.

For *symbolic* $(0, 1, X)$-simulation a new variable Z is introduced, which is used to model the unknown value X of the black box outputs. Now, for each output g_i of the incomplete design with primary input variables $x_1, \ldots, x_n$, a BDD representation of g_i is obtained by using a slightly modified version of symbolic simulation [7]. g_i depends on variables $x_1, \ldots, x_n$ and Z and has the following property:

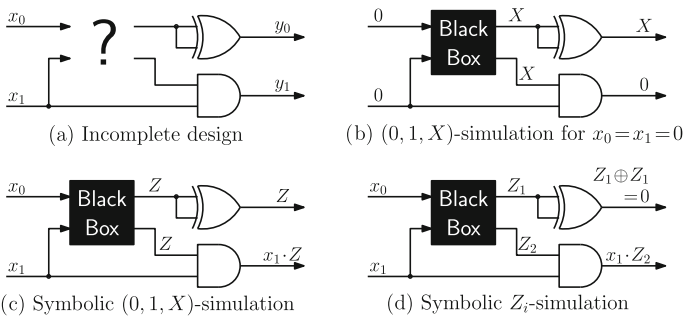

Fig. 2.8 Different methods to analyze an incomplete design

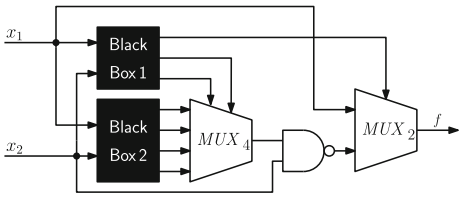

Fig. 2.9 An exemplary incomplete circuit

$$g_i|_{\substack{x_1=\varepsilon_1\\ \ldots\\ x_n=\varepsilon_n}} = \begin{cases} 1, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\varepsilon_1, \ldots, \varepsilon_n) \text{ produces } 1 \\ 0, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\varepsilon_1, \ldots, \varepsilon_n) \text{ produces } 0 \\ Z, \text{ if (0,1,}X\text{)-simulation with} \\ \quad \text{input } (\varepsilon_1, \ldots, \varepsilon_n) \text{ produces } X \end{cases} \tag{2.8}$$

See Fig. 2.8c for an example.

To compute BDDs for the functions g_i by symbolic simulation the inputs of the circuit are associated with unique BDD variables as in a conventional symbolic simulation [7]. All output signals of black boxes are associated with the new variable Z. Now BDDs for the functions computed by the gates of the circuit are built in topological order treating the black box outputs (associated with variable Z) like inputs of the circuit. The gates of the circuit can then be processed in a manner similar to a conventional symbolic simulation.[6] When we process an and_2 (or_2) gate, we combine the BDDs for the two predecessor functions by a BDD *AND* (*OR*) operation as in the conventional symbolic simulation. For an inv gate we perform a *NOT* operation on the BDD of the predecessor function, now followed by a *compose* operation (see, i.e., [6]) which composes $\overline{Z}$ for Z (written as $g|_{Z \leftarrow \overline{Z}}$ for a composition of $\overline{Z}$ for Z in g).

It is easy to see that this (modified) symbolic simulation leads to BDD representations fulfilling property (2.8).

In symbolic Z_i-simulation, a new variable Z_i is introduced for each black box output instead of using the same variable Z for all black box outputs, and a (conventional) symbolic simulation is performed. Figure 2.8d shows an example for symbolic Z_i-simulation. (Note that—in contrast to symbolic (0, 1, X)-simulation in Fig. 2.8c—the first output can now be proven to be constant 0.)

Flexible Z/Z_i-representation of incomplete circuits combines the two methods and allows some black box outputs to be represented as in symbolic (0, 1, X)-simulation and some black box outputs as in symbolic Z_i-simulation. Such a flexible representation of an incomplete circuit is computed as follows:

For each output of the black boxes, which are to be handled as in symbolic (0, 1, X)-simulation, we use the variable Z to model the black box output, while for each output of the black boxes, which are to be handled by symbolic Z_i-simulation we

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

use a new Z_i variable. The simulation now considers the latter black box outputs as additional inputs and then performs symbolic (0, 1, X)-simulation (always replacing Z by $\overline{Z}$ when processing inv gates).

It is easy to see that the resulting BDD representation for the circuit outputs g_i with primary input variables $x_1, \ldots, x_n$, (Z_i-simulated) black box outputs $Z_1, \ldots, Z_m$ and the Z-variable as inputs have the following property:

$$g_i|_{\substack{x_1=\varepsilon_1 \\ \cdots \\ x_n=\varepsilon_n \\ Z_1=\eta_1 \\ \cdots \\ Z_m=\eta_m}} = \begin{cases} 1, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\varepsilon_1, \ldots, \varepsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } 1 \\ 0, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\varepsilon_1, \ldots, \varepsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } 0 \\ Z, \text{ if (0,1,}X\text{)-simulation with input} \\ \quad (\varepsilon_1, \ldots, \varepsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } X. \end{cases}$$

Example 2.6 Figure 2.9 shows an example: If this circuit is simulated by using symbolic (0, 1, X)-simulation (meaning that Z is assigned to the outputs of both black box 1 and black box 2), a total number of 3 variables are needed (x_1, x_2, Z) and the resulting function for the output is $f_Z = Z$.

If the circuit is simulated by using symbolic Z_i-simulation instead (meaning that for each output of black box 1 and black box 2 a new Z_i variable is used), 9 variables are needed ($x_1, x_2, Z_1, \ldots, Z_7$), and the function for the output is $f_{Z_i} = \overline{Z}_1 x_1 + Z_1 \cdot \left(\overline{x}_2 + \neg(\overline{Z}_2 \overline{Z}_3 Z_4 + Z_2 \overline{Z}_3 Z_5 + \overline{Z}_2 Z_3 Z_6 + Z_2 Z_3 Z_7)\right)$ (when variables $Z_1, \ldots Z_7$ are assigned top down to the black box outputs appearing in Fig. 2.9).

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

So, the flexible method generates an output function that is obviously less complicated than the result of symbolic Z_i-simulation, yet contains more information than the result of symbolic (0, 1, X)-simulation. To give an example, for $x_1 = 1$ and $x_2 = 0$, the output can be proven to be 1 using the flexible method, while it is not possible to obtain this information from symbolic (0, 1, X)-simulation.

(2) Basic principle of symbolic model checking for incomplete designs

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

The situation becomes more complex if we consider incomplete circuits, since for each replacement of the black boxes we may have different state sets satisfying φ. In contrast to conventional model checking, we will consider two sets instead of $Sat(\varphi)$. These two sets result from different replacements of the black boxes in incomplete circuits:

Definition 2.5 Let I be an incomplete circuit and let r be a replacement of the black boxes in I by (combinational or sequential) circuits which transforms I into a complete circuit I^r. Let $\mathscr{R}$ be the set of all possible replacements of this kind. For a replacement $r \in \mathscr{R}$ let $Sat^r(\varphi)$ be the set of states of I^r satisfying φ. For the case that the replacement r contains *sequential* circuits, we define a set $Sat^r_E(\varphi)$ which results from $Sat^r(\varphi)$ by existentially quantifying the state bits corresponding to memory elements inside the black box replacements, and the set $Sat^r_A(\varphi)$ which results by universally quantifying these state bits.

- $Sat^{\text{exact}}_E(\varphi)$ is defined by $Sat^{\text{exact}}_E(\varphi) := \bigcup_{r\in\mathscr{R}} Sat^r_E(\varphi)$. We define that a state in $Sat^{\text{exact}}_E(\varphi)$ *possibly* satisfies the property φ.
- $Sat^{\text{exact}}_A(\varphi)$ is defined by $Sat^{\text{exact}}_A(\varphi) := \bigcap_{r\in\mathscr{R}} Sat^r_A(\varphi)$. We define that a state in $Sat^{\text{exact}}_A(\varphi)$ *definitely* satisfies the property φ.

$Sat^{\text{exact}}_E(\varphi)$ contains all states, for which *there is* at least one black box replacement (together with some initialization for the memory elements inside these replacements) so that φ is satisfied. In order to compute $Sat^{\text{exact}}_E(\varphi)$, we could *conceptually* consider all possible replacements $r \in \mathscr{R}$ of the black boxes, compute $Sat^r(\varphi)$ for each such replacement by conventional model checking, perform an existential quantification for the state bits of memory elements inside the black boxes, and determine $Sat^{\text{exact}}_E(\varphi)$ as the union of all sets $Sat^r_E(\varphi)$. (Of course, this approach is not feasible, since the set $\mathscr{R}$ may be large and is not even finite, if we allow replacements by sequential circuits).

In a similar manner, $Sat^{\text{exact}}_A(\varphi)$ contains all states, for which φ is satisfied for *all* possible black box replacements (regardless of the initial values for the memory elements inside black boxes).

Given $Sat^{\text{exact}}_E(\varphi)$ and $Sat^{\text{exact}}_A(\varphi)$, it is easy to prove validity and to falsify realizability for the incomplete circuit:

Lemma 2.1 *If all initial states are included in $Sat^{exact}_A(\varphi)$, φ is satisfied for all replacements of the black boxes ("φ is valid").*

If there is at least one initial state not belonging to $Sat^{exact}_E(\varphi)$, there is no replacement of the black boxes so that φ is satisfied for the resulting complete circuit ("φ is not realizable").

Proof If all initial states are included in $Sat^{\text{exact}}_A(\varphi)$, then for each replacement $r \in \mathscr{R}$ all initial states are included in $Sat^r_A(\varphi)$. This in turn means that all extensions of initial states by arbitrary initial values for memory elements inside the black boxes are included in $Sat^r(\varphi)$, i.e., φ is valid.

If there is at least one initial state $\mathbf{s}^0$ outside of $Sat^{\text{exact}}_E(\varphi)$, then $\mathbf{s}^0$ is not included in $Sat^r_E(\varphi)$ for all replacements $r \in \mathscr{R}$ of the black boxes. Thus, all extensions of $\mathbf{s}^0$ by initial values for memory elements inside the black boxes are not included in $Sat^r(\varphi)$. Since it is not possible to choose a replacement for the black boxes (together with some initialization to the memory elements of the black boxes) such that the initial state $\mathbf{s}^0$ satisfies φ, φ is not realizable.

Just as black boxes in incomplete circuits lead to states definitely satisfying φ on the one hand and states possibly satisfying φ on the other hand, there are also two types of transitions between states in an incomplete circuit:

- There are 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
- there are 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').

Formally, fixed and possible transitions are defined as follows:

Definition 2.6 Let I be an incomplete circuit with state set $\mathbb{B}^{|\mathbf{s}|}$ and set of inputs $\mathbb{B}^{|\mathbf{x}|}$. Let $r \in \mathscr{R}$ be a replacement of the black boxes leading to a complete circuit I^r and let $\mathbb{B}^{|\mathbf{sr}|}$ be the state space formed by $|\mathbf{sr}|$ state bits inside the black box replacements.

- The incomplete circuit has a *fixed* transition from state $(\mathbf{s}, \mathbf{x}) \in \mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|}$ to state $(\mathbf{s}', \mathbf{x}')\mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|}$, if for each replacement $r \in \mathscr{R}$ and all $\mathbf{sr}, \mathbf{sr}' \in \mathbb{B}^{|\mathbf{sr}|}$, there is a transition from $(\mathbf{s}, \mathbf{sr}, \mathbf{x})$ to $(\mathbf{s}', \mathbf{sr}', \mathbf{x}')$ in the Mealy automaton corresponding to I^r.
- The incomplete circuit has a *possible* transition from state $(\mathbf{s}, \mathbf{x}) \in \mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|}$ to state $(\mathbf{s}', \mathbf{x}')\mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|}$, if there is a replacement $r \in \mathscr{R}$ and there are $\mathbf{sr}, \mathbf{sr}' \in \mathbb{B}^{|\mathbf{sr}|}$, such that there is a transition from $(\mathbf{s}, \mathbf{sr}, \mathbf{x})$ to $(\mathbf{s}', \mathbf{sr}', \mathbf{x}')$ in the Mealy automaton corresponding to I^r.

Fixed and possible transitions can be used in order to compute states that possibly or definitely satisfy a property φ.

(3) Approximations

For reasons of efficiency, we do not compute the exact sets of fixed and possible transitions and we do not compute the exact sets $Sat_E^{\text{exact}}(\varphi)$ and $Sat_A^{\text{exact}}(\varphi)$.

Instead of $Sat_E^{\text{exact}}(\varphi)$ and $Sat_A^{\text{exact}}(\varphi)$ we compute *approximations* $Sat_E(\varphi)$ and $Sat_A(\varphi)$ of these sets. To be more precise, we will compute over-approximations $Sat_E(\varphi) \supseteq Sat_E^{\text{exact}}(\varphi)$ of $Sat_E^{\text{exact}}(\varphi)$ and under-approximations $Sat_A(\varphi) \subseteq Sat_A^{\text{exact}}(\varphi)$ of $Sat_A^{\text{exact}}(\varphi)$.

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

This argumentation results in the following lemma:

Lemma 2.2 *Let $Sat_A(\varphi)$ be an under-approximation of $Sat_A^{exact}(\varphi)$, $Sat_E(\varphi)$ an over-approximation of $Sat_E^{exact}(\varphi)$. If all initial states are included in $Sat_A(\varphi)$, then φ is valid. If there is an initial state that is not included in $Sat_E(\varphi)$, then φ is not realizable.*

Approximations $Sat_E(\varphi)$ and $Sat_A(\varphi)$ will be computed based on an approximate transition relation and on approximate output functions for the corresponding Mealy automaton M. Approximations of transition function δ and output function λ are computed using symbolic Z/Z_i-simulations as defined above.

We start with the computation of two approximations of the sets of states in which λ_i is true, i.e., we start with the computation of under-approximations $Sat_A(y_i)$ and overapproximations $Sat_E(y_i)$. Under-approximations $Sat_A(\varphi)$ and over-approximations $Sat_E(\varphi)$ for arbitrary CTL formulas φ will be considered later on in this section.

For an incomplete circuit, let there be a number of black boxes with outputs modeled by Z and some other black boxes with outputs modeled by Z_i's. We apply symbolic Z/Z_i-simulation for computing the transition functions and the output functions. Thus, we introduce new variables Z and $\mathbf{Z}_l = (Z_{l,1}, Z_{l,2}, \ldots)$. The symbolic symbolic Z/Z_i-simulation now provides symbolic representations of the output functions $\lambda_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l)$ and transition functions $\delta_j(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l)$.

In standard model checking for complete designs, an atomic property y_i is satisfied for a state $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}) \in \mathbb{B}^{|\mathbf{s}|\times|\mathbf{x}|}$ if $\lambda_i|_{\mathbf{s}=\mathbf{s}^{\text{fix}},\mathbf{x}=\mathbf{x}^{\text{fix}}} = 1$.

Here we include a state $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ into $Sat_A(y_i)$, if λ_i is 1 for $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ assigned to $(\mathbf{s}, \mathbf{x})$ and *all* possible assignments to Z and $\mathbf{Z}_l$. We include $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ into $Sat_E(y_i)$, if λ_i is 1 for $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ assigned to $(\mathbf{s}, \mathbf{x})$ and some assignment to Z and $\mathbf{Z}_l$. Thus we define the characteristic functions of $Sat_A(y_i)$ and $Sat_E(y_i)$ as follows:

Definition 2.7

$$\chi_{Sat_A(y_i)}(\mathbf{s}, \mathbf{x}) := \forall Z \forall \mathbf{Z}_l \big(\lambda_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l)\big) \tag{2.9}$$

$$\chi_{Sat_E(y_i)}(\mathbf{s}, \mathbf{x}) := \exists Z \exists \mathbf{Z}_l \big(\lambda_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l)\big) \tag{2.10}$$

Lemma 2.3 *For $Sat_A(y_i)$ and for $Sat_E(y_i)$ as defined in Definition 2.7:*

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

Proof If $\lambda_i|_{\mathbf{s}=\mathbf{s}^{\text{fix}},\mathbf{x}=\mathbf{x}^{\text{fix}}} = 1$ for some state $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}) \in \mathbb{B}^{|\mathbf{s}|\times|\mathbf{x}|}$, then we know that λ_i is 1 in this state independently from the replacement of the black boxes, so $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ can be included into $Sat_A(y_i)$ and $Sat_E(y_i)$.

If $\lambda_i|_{\mathbf{s}=\mathbf{s}^{\text{fix}},\mathbf{x}=\mathbf{x}^{\text{fix}}} = 0$, then the output λ_i is 0 in this state independently from the replacement of the black boxes, so we can include $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ neither into $Sat_A(y_i)$ nor into $Sat_E(y_i)$.

In any other case, the value of y_i is unknown in this state and thus we can include $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$.

By Eq. (2.9) only states $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ with $\lambda_i|_{\mathbf{s}=\mathbf{s}^{\text{fix}},\mathbf{x}=\mathbf{x}^{\text{fix}}} = 1$ are included in $Sat_A(y_i)$, and by Eq. (2.10) all states $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ with $\lambda_i|_{\mathbf{s}=\mathbf{s}^{\text{fix}},\mathbf{x}=\mathbf{x}^{\text{fix}}} \neq 0$ are included in $Sat_E(y_i)$.

Now the computation of $Sat_A(\varphi)$ and $Sat_E(\varphi)$ is performed based on fixed and possible transitions. Here we work with approximations, too: We compute an over-approximation of the possible transitions, represented by the characteristic function

$\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{s}')$. (An under-approximation $\chi_{R_A}(\mathbf{s}, \mathbf{x}, \mathbf{s}')$ containing at most the fixed transitions could be computed as well, however it turns out that it is not really needed for our algorithm.)

We define our over-approximation of the possible transitions as follows:

Definition 2.8

$$\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{s}') := \exists \mathbf{Z}_l \Big(\prod\nolimits_{i=0}^{|\mathbf{s}|-1} \exists Z \big(\delta_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l) \equiv s_i' \big) \Big). \tag{2.11}$$

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

Lemma 2.4 *If* $\chi_{R_E}(\mathbf{s}^{fix}, \mathbf{x}^{fix}, \mathbf{s'}^{fix}) = 0$*, then there is no possible transition from* $(\mathbf{s}^{fix}, \mathbf{x}^{fix})$ *to* $(\mathbf{s'}^{fix}, \mathbf{x'}^{fix})$ *(for an arbitrary next input* $\mathbf{x'}^{fix}$*).*

Proof If $\chi_{R_E}(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{s'}^{\text{fix}}) = 0$, then $\forall \mathbf{Z}_l \Big(\bigvee_{i=0}^{|\mathbf{s}|-1} \forall Z \big(\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l) \not\equiv s_i'^{\text{fix}} \big) \Big) = 1$. This means that for an arbitrary fixed output $\mathbf{Z}_l^{\text{fix}}$ of the black boxes modeled by Z_i's there is an $0 \leq i \leq |\mathbf{s}|-1$ with $\forall Z \big(\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l^{\text{fix}}) \not\equiv s_i'^{\text{fix}} \big) = 1 (\star)$. According to the symbolic Z-simulation, $\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l^{\text{fix}}) = Z$, $\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l^{\text{fix}}) = s_i'^{\text{fix}}$, or $\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l^{\text{fix}}) = \neg s_i'^{\text{fix}}$. In the two former cases, $(\star)$ would not hold, thus we have $\delta_i(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, Z, \mathbf{Z}_l^{\text{fix}}) = \neg s_i'^{\text{fix}}$, i.e., the output value of δ_i differs from $s_i'^{\text{fix}}$ independently from the behavior of the Z-modeled black boxes.

Altogether we can conclude that the output value of δ for input $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ differs from $\mathbf{s'}^{\text{fix}}$ independently from the values at the outputs of black boxes, i.e., there cannot be a possible transition from $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}})$ to $(\mathbf{s'}^{\text{fix}}, \mathbf{x'}^{\text{fix}})$.

Remark 2.1 Extending Definitions 2.5 and 2.6 with respect to our approximations, we will denote in the following not only the states in $Sat_E^{\text{exact}}(\varphi)$, but also the states in $Sat_E(\varphi)$ by 'states *possibly* satisfying φ.' Similarly, we characterize all transitions described by χ_{R_E} as 'possible transitions.'

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

For each state $(\mathbf{s}, \mathbf{x})$, $\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{s}')$ gives us the set of $\mathbf{s}'$values the possible successors can have. Each of these different $\mathbf{s}'$values represents a *set* $S_{\mathbf{s}'} := \{(\mathbf{s}', \mathbf{x}') | \mathbf{x}' \in \mathbb{B}^{|\mathbf{x}|}\}$ of possible successor states sharing this $\mathbf{s}'$ value (yet with arbitrary value of $\mathbf{x}'$). So, if for a state $(\mathbf{s}, \mathbf{x})$ one of the states in one of these possible successor sets $S_{\mathbf{s}'}$ possibly satisfies ψ (i.e., is in $Sat_E(\psi)$), the current state possibly satisfies $EX\psi$ and can thus be included in $Sat_E(EX\psi)$. Figure 2.10 illustrates the sets.

Definition 2.9 $Sat_E(EX\psi)$ is the set of states for which there is a possible successor that is in $Sat_E(\psi)$. This is represented by:

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

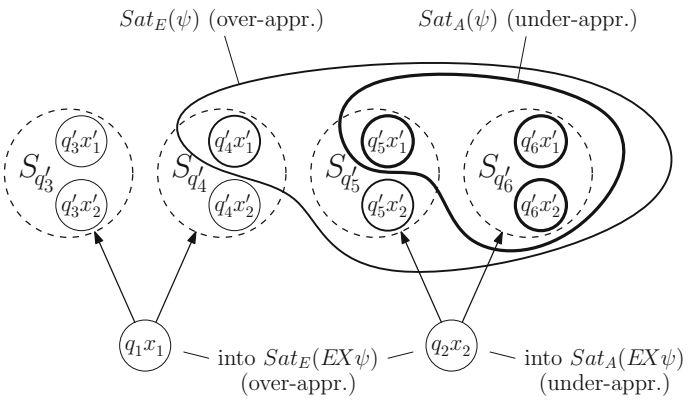

Fig. 2.10 Evaluation of $Sat_A(EX\psi)$, $Sat_E(EX\psi)$

On the other hand, if in each set $S_{\mathbf{s}'}$ of possible successors of $(\mathbf{s}, \mathbf{x})$ there is at least one state that definitely satisfies ψ (i.e., is in $Sat_A(\psi)$), then for each black box implementation at least one successor of state $(\mathbf{s}, \mathbf{x})$ satisfies ψ and thus, the current state $(\mathbf{s}, \mathbf{x})$ definitely satisfies $EX\psi$ and can be included in $Sat_A(EX\psi)$. Again, Fig. 2.10 illustrates the sets.

Definition 2.10 $Sat_A(EX\psi)$ is the set of states for which in each set $S_{\mathbf{s}'}$ of possible successors there is at least one state that is in $Sat_A(\psi)$. This is represented by:

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

Lemma 2.5 *Let $Sat_A(\psi)$ be an under-approximation of $Sat_A^{exact}(\psi)$ and $Sat_E(\psi)$ be an over-approximation of $Sat_E^{exact}(\psi)$. For $Sat_A(EX\psi)$ as defined in Definition 2.10 and for $Sat_E(EX\psi)$ as defined in Definition 2.9, the following holds:*

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

The proof of this lemma follows from the considerations given above the definitions.

Negation can be defined as follows: Since $Sat_E(\psi)$ is an over-approximation of all states in which ψ *may be* satisfied for some black box replacement, we do know that for an arbitrary state in $\mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|} \backslash Sat_E(\psi)$ 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 $\mathbb{B}^{|\mathbf{s}|} \times \mathbb{B}^{|\mathbf{x}|} \backslash Sat_E(\psi)$ as an under-approximation $Sat_A(\neg\psi)$. Since an analogous argument holds for $Sat_A(\psi)$ and $Sat_E(\neg\psi)$ we define

Definition 2.11 $\chi_{Sat_A(\neg\psi)}(\mathbf{s}, \mathbf{x}) := \overline{\chi_{Sat_E(\psi)}(\mathbf{s}, \mathbf{x})}$ and $\chi_{Sat_E(\neg\psi)}(\mathbf{s}, \mathbf{x}) := \overline{\chi_{Sat_A(\psi)}(\mathbf{s}, \mathbf{x})}$.

$Sat_A(\psi_1 \vee \psi_2)$ and $Sat_E(\psi_1 \vee \psi_2)$ are computed as usual in model checking for complete designs.

Definition 2.12 $\chi_{Sat_A(\psi_1 \vee \psi_2)}(\mathbf{s}, \mathbf{x}) := (\chi_{Sat_A(\psi_1)} \vee \chi_{Sat_A(\psi_2)})(\mathbf{s}, \mathbf{x})$ and $\chi_{Sat_E(\psi_1 \vee \psi_2)}(\mathbf{s}, \mathbf{x}) := (\chi_{Sat_E(\psi_1)} \vee \chi_{Sat_E(\psi_2)})(\mathbf{s}, \mathbf{x})$.

Finally, we define $\varphi = EG\psi$ and $\varphi = E\psi_1 U \psi_2$ to be evaluated by their standard fixed point iterations (see Fig. 2.7) based on the evaluation of EX defined above (two separate fixed point iterations for Sat_A and Sat_E). We do not need to define more CTL operations, since other CTL operations can be expressed using the operations discussed so far.

Lemma 2.6 *For the sets* $Sat_A(\neg\psi)$, $Sat_E(\neg\psi)$, $Sat_A(\psi_1 \vee \psi_2)$, $Sat_E(\psi_1 \vee \psi_2)$, $Sat_A(EG\psi)$, $Sat_E(EG\psi)$, $Sat_A(E\psi_1 U \psi_2)$, *and* $Sat_E(E\psi_1 U \psi_2)$ *as defined above, the following holds:*

$$\begin{aligned}
Sat_A(\neg\psi) &\subseteq Sat_A^{exact}(\neg\psi),\\
Sat_E^{exact}(\neg\psi) &\subseteq Sat_E(\neg\psi),\\
Sat_A(\psi_1 \vee \psi_2) &\subseteq Sat_A^{exact}(\psi_1 \vee \psi_2),\\
Sat_E^{exact}(\psi_1 \vee \psi_2) &\subseteq Sat_E(\psi_1 \vee \psi_2),\\
Sat_A(EG\psi) &\subseteq Sat_A^{exact}(EG\psi),\\
Sat_E^{exact}(EG\psi) &\subseteq Sat_E(EG\psi),\\
Sat_A(E\psi_1 U \psi_2) &\subseteq Sat_A^{exact}(E\psi_1 U \psi_2),\\
Sat_E^{exact}(E\psi_1 U \psi_2) &\subseteq Sat_E(E\psi_1 U \psi_2).
\end{aligned}$$

Theorem 2.1 *The result of the recursive computation can be evaluated as follows*[7]*:*

$$\begin{aligned}
\left(\forall \mathbf{x}(\chi_{Sat_A(\varphi)}|_{\mathbf{s}=\mathbf{s}^0})\right) = 1 \quad &\Longrightarrow \varphi \textit{ is valid}\\
\left(\exists \mathbf{x}(\overline{\chi_{Sat_E(\varphi)}}|_{\mathbf{s}=\mathbf{s}^0})\right) = 1 &\Longrightarrow \varphi \textit{ is not realizable}.
\end{aligned}$$

Proof The proof follows directly from Lemmas 2.2, 2.3, 2.5, and 2.6.

(4) Including Z_i-variables into the state space

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

As a motivation for this, consider the simple CTL formula $EF(y \wedge \neg y)$ for a design in which a black box output is directly connected to the primary output y. In every state, $(\mathbf{s}, \mathbf{x})$ both y and $\neg y$ are *possibly* satisfied (depending on the black box implementation), but they are not *definitely* satisfied. Thus, the method described so far computes the result that $y \wedge \neg y$ is possibly satisfied in every state $(\mathbf{s}, \mathbf{x})$, but not definitely, and the same result holds for $EF(y \wedge \neg y)$. For this reason, the method is neither able to prove validity nor to falsify realizability for the given incomplete design and the given formula.

However, it is clear that there will be no point in time during the computation where y is simultaneously true and false. Problems of this kind can be solved if

[7] Remember that $\mathbf{s}^0$ is the initial state of the circuit.

we include Z_i-variables assigned to black box outputs into the states of the Kripke structure. In this way, the according black box output values Z_i are constant within each single state and therefore in our example y has a fixed value for each state.

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

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

Both the output function $\lambda(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l, \mathbf{Z}_o)$ and the transition function $\delta(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l, \mathbf{Z}_o)$ can be computed by using symbolic Z/Z_i-simulation, where for symbolic simulation it is not necessary to distinguish between $\mathbf{Z}_l$ and $\mathbf{Z}_o$.

The computation of sets $Sat_A(\cdot)$ and $Sat_E(\cdot)$ is performed in a manner similar to the previous section. We start with the sets of states definitely or possibly satisfying the atomic CTL formula y_i:

We include a state $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}}) \in \mathbb{B}^{|\mathbf{s}|\times|\mathbf{x}|\times|\mathbf{Z}_o|}$ into $Sat_A(y_i)$ (and $Sat_E(y_i)$), if λ_i is 1 for $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}})$ assigned to $(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o)$ and *all* possible assignments to Z and $\mathbf{Z}_l$, since in this case λ_i is 1 in this state independently from the replacement of the black boxes.

We include $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}}) \in \mathbb{B}^{|\mathbf{s}|\times|\mathbf{x}|\times|\mathbf{Z}_o|}$ into $Sat_E(y_i)$, if λ_i is 1 for $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}})$ assigned to $(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o)$ and *some* assignment to Z and $\mathbf{Z}_l$. However, if λ_i is 0 for $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}})$ assigned to $(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o)$ and *all* assignments to Z and $\mathbf{Z}_l$, we include $(\mathbf{s}^{\text{fix}}, \mathbf{x}^{\text{fix}}, \mathbf{Z}_o^{\text{fix}})$ neither into $Sat_A(y_i)$ nor $Sat_E(y_i)$, since then the output λ_i is 0 in this state independently from the replacement of the black boxes. Altogether we define the characteristic functions of $Sat_A(y_i)$ and $Sat_E(y_i)$ as

Definition 2.13

$$\chi_{Sat_A(y_i)}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o) := \forall Z \forall \mathbf{Z}_l \left(\lambda_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l, \mathbf{Z}_o)\right)$$
$$\chi_{Sat_E(y_i)}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o) := \exists Z \exists \mathbf{Z}_l \left(\lambda_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l, \mathbf{Z}_o)\right).$$

Lemma 2.7 *For $Sat_A(y_i)$ and for $Sat_E(y_i)$ as defined in Definition 2.13:*

$$\forall \mathbf{Z}_o \chi_{Sat_A(y_i)} \leq \chi_{Sat_A^{exact}(y_i)}, \quad \chi_{Sat_E^{exact}(y_i)} \leq \exists \mathbf{Z}_o \chi_{Sat_E(y_i)}.$$

The lemma follows from the considerations given above the definition.

Analogously, we define an over-approximation χ_{R_E} for the characteristic function of possible transitions:

Definition 2.14

$$\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o, \mathbf{s}') := \left(\exists \mathbf{Z}_l \prod_{i=0}^{|\mathbf{s}|-1} \exists Z \left(\delta_i(\mathbf{s}, \mathbf{x}, Z, \mathbf{Z}_l, \mathbf{Z}_o) \equiv s_i'\right)\right).$$

Based on $Sat_A(y_i)$, $Sat_E(y_i)$ and χ_{R_E}, the sets $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$ can be computed by arguments similar to the previous section. The main difference is that we have to handle the additional variables $\mathbf{Z}_o$ in the state space correctly:

$\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o, \mathbf{s}')$ gives us the $\mathbf{s}'$ values of possible successors of a state $(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o)$. Now each of these different $\mathbf{s}'$ values represents a set $S_{\mathbf{s}'} := \{(\mathbf{s}', \mathbf{x}', \mathbf{Z}'_o) \mid \mathbf{x}' \in \mathbb{B}^{|\mathbf{x}|}, \mathbf{Z}'_o \in \mathbb{B}^{|\mathbf{Z_o}|}\}$ of possible successor states sharing this $\mathbf{s}'$ value. For $Sat_E(EX\psi)$, we include all states $(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o)$ with the following property: There exists a possible successor set $S_{\mathbf{s}'}$ in which there is a $\mathbf{x}'$, so that for at least one black box output value $\mathbf{Z}'_o$: $(\mathbf{s}', \mathbf{x}', \mathbf{Z}'_o)$ possibly satisfies ψ.

Definition 2.15 We define $Sat_E(EX\psi)$ by

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

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

Definition 2.16 We define $Sat_A(EX\psi)$ by

$$\chi_{Sat_A(EX\psi)}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o) := \forall \mathbf{s}' \Big(\chi_{R_E}(\mathbf{s}, \mathbf{x}, \mathbf{Z}_o, \mathbf{s}') \rightarrow \exists \mathbf{x}' \forall \mathbf{Z}'_o \big(\chi_{Sat_A(\psi)} \big|^{\mathbf{s} \leftarrow \mathbf{s}'}_{\substack{\mathbf{x} \leftarrow \mathbf{x}' \\ \mathbf{Z}_o \leftarrow \mathbf{Z}'_o}} \big)(\mathbf{s}', \mathbf{x}', \mathbf{Z}'_o) \Big).$$

Lemma 2.8 *Let $\forall \mathbf{Z}_o \chi_{Sat_A(\psi)}$ represent an under-approximation of $Sat_A^{exact}(\psi)$ and let $\exists \mathbf{Z}_o \chi_{Sat_E(\psi)}$ represent an over-approximation of $Sat_E^{exact}(\psi)$. For $Sat_A(EX\psi)$ as defined in Definition 2.16 and for $Sat_E(EX\psi)$ as defined in Definition 2.15, the following holds:*

$$\forall \mathbf{Z}_o \chi_{Sat_A(EX\psi)} \leq \chi_{Sat_A^{exact}(EX\psi)} \text{ and } \chi_{Sat_E^{exact}(EX\psi)} \leq \exists \mathbf{Z}_o \chi_{Sat_E(EX\psi)}.$$

Again, the proof of the lemma follows from the considerations given above the definitions.

The computation of all remaining CTL operators $\neg$, EG and EU is performed as already described before.

Theorem 2.2 *The result of the recursive computation can be evaluated as follows:*

$$\begin{aligned} \big(\forall \mathbf{x} \forall \mathbf{Z}_o (\chi_{Sat_A(\varphi)}|_{\mathbf{s}=\mathbf{s}^0})\big) = 1 \quad &\Longrightarrow \varphi \text{ is valid} \\ \big(\exists \mathbf{x} \forall \mathbf{Z}_o (\overline{\chi_{Sat_E(\varphi)}}|_{\mathbf{s}=\mathbf{s}^0})\big) = 1 &\Longrightarrow \varphi \text{ is not realizable.} \end{aligned}$$

Proof The proof follows from Lemmas 2.2, 2.7, and 2.8.

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

(5) Exact symbolic model checking for black boxes with bounded memory
Despite the methods described so far being approximate, they are able to disprove realizability and prove validity in many practically relevant cases [24]. As already mentioned in Sect. 2.4, the general decision problem with several black boxes is undecidable [26] however. [24] presents a concept how to provide an *exact* solution to a restricted problem by means of a *conventional symbolic model checker*. Under assumption of an upper bound to the number of the internal states of the black boxes ('bounded memory'), the exact set of black box replacements for which a property is satisfied can be symbolically computed, i.e., an exact answer to both the realizability and the validity question can be given under the bounded memory assumption. 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.

2.5 Conclusion

In this chapter, we have provided a comprehensive study of the verification of incomplete combinational and sequential circuits. We have presented different methods for modeling black boxes: (1) 01X-logic, which is efficient, but pessimistic and overestimates the set of signals in the circuit which carry an unknown value, (2) quantified Boolean formulas, which constitute an exact formalism for combinational circuits with a single black box, but which are incomplete in other cases, and (3) dependency-quantified Boolean formulas, which are accurate for both combinational and sequential circuits with an arbitrary number of black boxes as long as an upper bound on the amount of memory in the black boxes can be given in advance. We have provided algorithms trading off complexity and precision for (1) the partial equivalence checking (PEC) problem of combinational circuits and (2) the realizability (validity) of invariant properties and general CTL properties for incomplete sequential circuits. For invariant properties we presented algorithms based on SAT, QBF and DQBF solvers, for CTL properties algorithms based on symbolic BDD or AIG representations. If the problems are proven to be realizable by our DQBF based method, our solver HQS is also able to extract corresponding implementations for the black boxes as Skolem functions for the existential variables [31]. An interesting open problem is the question how to compute certificates for *unrealizability* as well for checking with a separate proof checker. Another objective for future work is further increasing the efficiency of the underlying SAT, QBF, and DQBF solvers to improve the scalability of the approach for the approximative as well as the exact methods. Moreover, it will be interesting to look into the generalization of other successful verification methods such as Property Directed Reachability (PDR or IC^3) [5, 11] from complete to incomplete circuits.

References

1. A. Biere, Resolve and expand, in *International Conference on Theory and Applications of Satisfiability Testing (SAT), Vancouver, BC, Canada* (2004)
2. A. Biere, M. Heule, H. van Maaren, T. Walsh (ed.), in *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, vol. 185 (IOS Press, 2008)
3. R. Bloem, U. Egly, P. Klampfl, R. Könighofer, F. Lonsing, SAT-based methods for circuit synthesis, in *International Conference on Formal Methods in Computer Aided Design (FMCAD), Lausanne, Switzerland* (IEEE, 2014), pp. 31–34
4. R. Bloem, R. Könighofer, M. Seidl, SAT-based synthesis methods for safety specs, in *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, ed. By K.L. McMillan, X. Rival. LNCS, vol. 8318 (Springer, San Diego, CA, USA, 2014), pp. 1–20
5. A.R. Bradley, SAT-based model checking without unrolling, in *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. LNCS, vol. 6538 (Springer, 2011), pp. 70–87
6. R.E. Bryant, Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. Aided Des. **35**(8), 677–691 (1986)
7. R.E. Bryant, Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Comput. Surv. **24**, 293–318 (1992)
8. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**(2), 142–170 (1992)
9. E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
10. S.A. Cook, The complexity of theorem-proving procedures, in *Annual ACM Symposium on Theory of Computing (STOC)* (ACM Press, 1971), pp. 151–158
11. N. Eén, A. Mishchenko, R.K. Brayton, Efficient implementation of property directed reachability, in *International Conference on Formal Methods in Computer Aided Design (FMCAD)* (FMCAD Inc., 2011), pp. 125–134
12. A. Fröhlich, G. Kovásznai, A. Biere, H. Veith, iDQ: Instantiation-based DQBF solving, in *International Workshop on Pragmatics of SAT (POS)*, ed. By D.L. Berre. EPiC Series, vol. 27 (EasyChair, Vienna, Austria, 2014), pp. 103–116
13. K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, B. Becker, Equivalence checking for partial implementations revisited, in *Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV)*, ed. By C. Haubelt, D. Timmermann (Universität Rostock, ITMZ, Rostock, Germany, 2013), pp. 61–70
14. K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, B. Becker, Equivalence checking of partial designs using dependency quantified Boolean formulae, in *IEEE International Conference on Computer Design (ICCD), Asheville, NC, USA* (IEEE Computer Society, 2013), pp. 396–403
15. K. Gitina, R. Wimmer, S. Reimer, M. Sauer, C. Scholl, B. Becker, Solving DQBF through quantifier elimination, in *International Conference on Design, Automation and Test in Europe (DATE), Grenoble, France* (IEEE, 2015)
16. E. Giunchiglia, P. Marin, M. Narizzano, sQueezeBF: an effective preprocessor for QBFs based on equivalence reasoning, in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, ed. By O. Strichman, S. Szeider. LNCS, vol. 6175 (Springer, Edinburgh, UK, 2010), pp. 85–98
17. A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, M.S. Hsiao, Testing, verification, and diagnosis in the presence of unknowns, in *IEEE VLSI Test Symposium (VTS)* (IEEE Computer Society, Montreal, Canada, 2000), pp. 263–270
18. S. Jo, A.M. Gharehbaghi, T. Matsumoto, M. Fujita, Debugging processors with advanced features by reprogramming LUTs on FPGA, in *International Conference on Field-Programmable Technology (FPT)* (IEEE, Kyoto, Japan, 2013), pp. 50–57

19. S. Jo, T. Matsumoto, M. Fujita, Sat-based automatic rectification and debugging of combinational circuits with LUT insertions. IPSJ Trans. Syst. LSI Des. Methodol. **7**, 46–55 (2014)
20. K.L. McMillan, *Symbolic Model Checking* (Kluwer Academic Publisher, 1993)
21. A.R. Meyer, L.J. Stockmeyer, Word problems requiring exponential time: preliminary report, in *Annual ACM Symposium on Theory of Computing (STOC)* (ACM Press, 1973), pp. 1–9
22. C. Miller, S. Kupferschmid, M.D.T. Lewis, B. Becker, Encoding techniques, Craig interpolants and bounded model checking for incomplete designs, in *International Conference on Theory and Applications of Satisfiability Testing (SAT)*. LNCS, vol. 6175 (Springer, 2010), pp. 194–208
23. C. Miller, C. Scholl, B. Becker, Proving QBF-hardness in bounded model checking for incomplete designs, in *International Workshop on Microprocessor Test and Verification (MTV)* (IEEE Computer Society, Austin, TX, USA, 2013)
24. T. Nopper, C. Scholl, Symbolic model checking for incomplete designs with flexible modeling of unknowns. IEEE Trans. Comput. **62**(6), 1234–1254 (2013)
25. G. Peterson, J. Reif, S. Azhar, Lower bounds for multiplayer non-cooperative games of incomplete information. Comput. Math. Appl. **41**(7–8), 957–992 (2001)
26. A. Pnueli, R. Rosner, Distributed systems are hard to synthesize, in *IEEE Symposium on Foundations of Computer, Science* (1990), pp. 746–757
27. C. Scholl, B. Becker, Checking equivalence for partial implementations, in *ACM/IEEE Design Automation Conference (DAC)* (ACM Press, Las Vegas, NV, USA, 2001), pp. 238–243
28. A. Smith, A.G. Veneris, M.F. Ali, A. Viglas, Fault diagnosis and logic debugging using boolean satisfiability. IEEE Trans. CAD Integr. Circuits Syst. **24**(10), 1606–1621 (2005)
29. A. Sülflow, G. Fey, R. Drechsler, Using QBF to increase accuracy of SAT-based debugging, in *International Symposium on Circuits and Systems (ISCAS)* (IEEE, Paris, France, 2010), pp. 641–644
30. G.S. Tseitin, On the complexity of derivation in propositional calculus. Stud. Constr. Math. Math. Logic Part **2**, 115–125 (1970)
31. K. Wimmer, R. Wimmer, C. Scholl, B. Becker, Skolem functions for DQBF, in *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, ed. By C. Artho, A. Legay, D. Peled. LNCS, vol. 9938 (Springer, Chiba, Japan, 2016)

Author Biographies

Bernd Becker received the Diploma in Mathematics in 1979, the Doctoral and the Habilitation degree in Computer Science in 1982 and 1988, respectively, all from Saarland University. In 1989, he joined the Institut für Informatik at J.W.Goethe- University Frankfurt as an Associate Professor for "Complexity Theory and Efficient Algorithms". Since 1995, he is a Full Professor (Chair of Computer Architecture) at the Faculty of Engineering, University of Freiburg. His research activities include design, test and verification methods for embedded systems and nanoelectronic circuitry. Bernd Becker was a Co-Speaker of the DFG Transregional Collaborative Research Center "Automatic Analysis and Verification of Complex Systems (AVACS)" from 2004 to 2015 and is a Director of the Centre for Security and Society, University of Freiburg. He is a fellow of IEEE and Member of Academia Europaea.

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

Ralf Wimmer received his diploma with distinction in computer science from the Albert-Ludwigs-Universität Freiburg, Germany in 2004. Afterwards, he worked as a Ph.D. student at the Chair of Computer Architecture at the same university, advised by Prof. Dr. Bernd Becker. He obtained his Ph.D. degree with distinction in 2011 for his thesis on symbolic methods for probabilistic verification. Since then, he is continuing his work as a research assistant and leader of the verification group at the Chair of Computer Architecture. His research focus is on symbolic methods and solver technologies, and their application for the verification of digital and stochastic systems.

http://www.springer.com/978-3-319-57683-1

Formal System Verification
State-of the-Art and Future Trends
Drechsler, R. (Ed.)
2018, XVI, 182 p. 71 illus., 49 illus. in color., Hardcover
ISBN: 978-3-319-57683-1