Preprint from Proceedings of GI/ITG/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Berlin, Germany, March 2009

Symbolic CTL Model Checking for Incomplete Designs by Selecting Property-Specific Subsets of Local Component Assumptions

Christian Miller   Tobias Nopper   Christoph Scholl

{millerc, nopper, scholl}@informatik.uni-freiburg.de

Institute of Computer Science, Albert-Ludwigs-University, D-79110 Freiburg, Germany

Black Box symbolic model checking is a method to check whether an incompletely specified circuit, in which some parts of the design have been replaced by 'Black Boxes', satisfies a CTL property regardless of the actual replacement of the Black Boxes. One possible application is model checking with abstraction where complex parts of the design (which are not really relevant for the property at hand) are put into Black Boxes in order to speed up the model checking run. However, due to its approximative nature, symbolic model checking for incomplete designs may be unable to provide proofs when the approximations are to coarse.

In this paper we show how Black Box model checking can profit from information about the Black-Boxed parts given in form of so-called local component assumptions. The assumptions we use in this context are safety properties arguing about fixed time windows. Furthermore we will present a set of heuristic-based approaches to automatically select a subset of local component assumptions that is sufficient to prove or disprove a given CTL property.

1 Introduction

Model checking [1, 2] is an important method for verifying whether a sequential circuit satisfies a given temporal property. *Symbolic* model checking (SMC) [3, 4] makes use of Binary Decision Diagrams (BDDs) [5], which are often able to represent state sets and state traversals that occur in model checking in an efficient form.

While the circuit design under verification is typically considered to be complete, in this work we focus on *incomplete* designs for which some parts of the circuit design are not yet implemented (so-called Black Boxes). There are many applications for model checking incomplete designs, such as early verification checks on unfinished designs, error localization in faulty designs and the abstraction of complex parts of a design in order to simplify the model checking task. Symbolic model checking for incomplete designs [6] is able to provide proofs that a given property is satisfied for all replacements of the Black Boxes ('validity') and proofs that the property is not satisfied for any replacement of the Black Boxes ('unrealizability').

However, it is possible that model checking of incomplete designs is unable to provide a result, since a property is neither valid nor unrealizable (i.e. it depends on the Black Box implementation whether the property holds or not) or since approximations during model checking prevent a proof of validity or unrealizability.

Assuming that the behavior of the Black Boxes is not completely unknown, we show how to specify properties describing the behavior of the Black Boxes (so-called *component assumptions*) and provide a method to include them into the model checking algorithm. Depending on the number and complexity of such assumptions, including *all* of them into model checking might be very time consuming. Therefore we provide heuristics to select appropriate assumptions by means of analyzing possible counterexamples in a CEGAR-like [7] fashion for ACTL formulas. Additionally, we will give some ideas how to heuristically select local component properties for arbitrary CTL formulas.

1.1 Related Work

The task to perform model checking when some components of the design are only described by local component properties, could also be solved by synthesizing replacements of these components and model checking the design that has been completed with these synthesized components.

The work in [8] comes closest to our approach. Based on local properties (CTL formulas without the next operator) that are given for components, an 'abstract Kripke structure' is build for each component and composed with the given design; nondeterminism is resolved by adding free primary inputs. Based on a CEGAR routine, local properties are added until a global ACTL formula can be proven.

An alternative possibility to synthesize components is presented in [9]. Based on fixed-time PSL descriptions, so-called ‘cando-objects’ are generated that can be used to complete the design for model checking. As in [8], the nondeterministic behavior of the components is modeled by additional primary inputs, so that only ACTL formulas can be proven (actually, only fixed-time safety properties are considered in [9]).

Both approaches listed above are limited to provide proofs for validity of ACTL formulas; in contrast to that, our method is able to process full CTL and to provide both proofs for unrealizability and for validity. We will show that our CEGAR-based heuristics to incrementally take local properties into consideration are effective for ACTL formulas, but we also provide some ideas for heuristics that can be applied for arbitrary CTL formulas.

Additionally, we are able to include the local component properties that are given for the Black Boxes directly in the model checking routine without the need to synthesize actual replacements of the Black Boxes.

2 Preliminaries

In this section we give a brief introduction into symbolic model checking for incomplete designs according to [6]. The introduction is illustrated by a small sequential example with one Black Box as given in Fig. 1. In order to keep things simple this example does not have any primary input.

2.1 Modeling of Unknowns

For symbolic CTL model checking of a given design, a symbolic representation of its transition function δ is 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 δ. [10, 6] introduced two different methods, modeling the Black Box outputs with differing accuracy: Symbolic $(0, 1, X)$-simulation (which is based on the well-known $(0, 1, X)$-simulation [11] and symbolic Z_i-simulation.[1] In this paper we consider only symbolic Z_i-simulation, but we plan to extend the methods wrt. symbolic $(0, 1, X)$-simulation in the future.

Symbolic Z_i-simulation introduces a new Z_i variable for each Black Box output and then performs symbolic simulation as if the Black Box outputs were inputs. For the example given in Fig. 1, this results in the following transition function (q is the boolean variable representing the value of the flipflop, Z represents the output of the Black Box; there is no primary input x in this design):

$$\delta(\vec{q}, \vec{x}, \vec{Z}) = \overline{q} + \overline{Z}$$

In the variant of model checking considered in this paper, we include Z_i variables into the state space, resulting in a state space over variables $(\vec{q}, \vec{x}, \vec{Z})$.

2.2 Possible Transitions

Based on the transition function δ, it is possible to compute the set of transitions between the states in the state space. Since for each state $(\vec{q}, \vec{x}, \vec{Z})$, both the value of the primary inputs $\vec{x}\,'$and the value of the Black Box outputs $\vec{Z}\,'$ in the succeeding state $(\vec{q}\,', \vec{x}\,', \vec{Z}\,')$ are not known, we consider transitions to all possible values of $\vec{x}\,'$and all possible values of $\vec{Z}\,'$ as *possible* transitions.

For our example in Fig. 1, the set of possible transitions (illustrated in Fig. 2) is as follows[2] :

$$\begin{aligned} R_E = \{ & ((\overline{q}, \overline{Z}), (q', \overline{Z'})), ((\overline{q}, \overline{Z}), (q', Z')), ((\overline{q}, Z), (q', \overline{Z'})), ((\overline{q}, Z), (q', Z')), \\ & ((q, \overline{Z}), (q', \overline{Z'})), ((q, \overline{Z}), (q', Z')), ((q, Z), (\overline{q}', \overline{Z'})), ((q, Z), (\overline{q}', Z')) \} \end{aligned}$$

Based on the set of possible transitions, model checking for incomplete designs recursively computes two sets for each partial CTL formula φ: $Sat_A(\varphi)$ underapproximates the set of states that satisfy φ *for*

[1]It turned out that model checking for incomplete designs using symbolic $(0, 1, X)$-simulation is typically much less complex than model checking using symbolic Z_i-simulation. On the other hand, the accuracy of approximations is higher with symbolic Z_i-simulation.

[2]To point out the connection of values to state variables we write $(\overline{q}, Z)$ for state $(0, 1)$ where 0 is assigned to state variable q, 1 is assigned to variable Z, and analogously for other valuations.

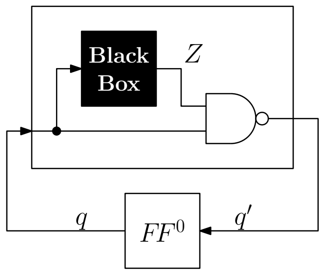


Figure 1: An exemplary sequential circuit with one Black Box. Initially, $q\!=\!0$.

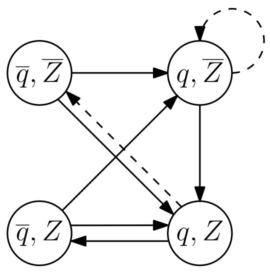


Figure 2: The possible transitions of the circuit in Fig. 1 (both solid and dashed arrows)

each replacement of the Black Boxes ('surely satisfy φ'), and $Sat_E(\varphi)$ overapproximates the set of states that satisfy φ *for at least one* replacement of the Black Boxes ('possibly satisfy φ').

Given these two sets $Sat_A(\varphi)$ and $Sat_E(\varphi)$, it is possible to provide approximate, yet sound proofs that either the property holds for every possible replacement of the Black Boxes ('property φ is valid') or that it is violated for every possible replacement of the Black Boxes ('property φ is not realizable'): If all initial states surely satisfy φ (i.e. lie within $Sat_A(\varphi)$), then the validity of φ is proven; if there is an initial state that does not even possibly satisfy φ (i.e. it lies outside $Sat_E(\varphi)$), then φ is unrealizable.

For a given state $(\vec{q},\vec{x},\vec{Z})$, the transition function δ computes the values of the flipflops $\vec{q}\,'$ in the next state $(\vec{q}\,',\vec{x}\,',\vec{Z}\,')$. The value $\vec{q}\,'$ defines a set $S_{\vec{q}\,'} := \{(\vec{q}\,',\vec{x}\,',\vec{Z}\,') \mid \vec{x}\,' \in \mathbb{B}^{|\vec{x}|}, \vec{Z}\,' \in \mathbb{B}^{|\vec{Z}|}\}$ of possible successor states sharing this $\vec{q}\,'$ value. For $Sat_E(EX\varphi)$, we include all states $(\vec{q},\vec{x},\vec{Z})$ with the following property: There exists a possible successor $(\vec{q}\,',\vec{x}\,',\vec{Z}\,')$ in $S_{\vec{q}\,'}$ which possibly satisfies φ. This leads to:

$$Sat_E(EX\varphi) := \big\{(\vec{q},\vec{x},\vec{Z}) \;\big|\; \exists\vec{q}\,'\exists\vec{x}\,'\exists\vec{Z}\,'\colon \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}\,',\vec{x}\,',\vec{Z}\,')\big) \in R_E \wedge (\vec{q}\,',\vec{x}\,',\vec{Z}\,') \in Sat_E(\varphi)\big\}$$

Similarly, for $Sat_A(EX\varphi)$, we include all states $(\vec{q},\vec{x},\vec{Z})$, for which in the set of possible successors $S_{\vec{q}\,'}$ there is a $\vec{x}\,'$, so that for all Black Box output values $\vec{Z}\,'$: $(\vec{q}\,',\vec{x}\,',\vec{Z}\,')$ definitely satisfies φ, i.e.

$$Sat_A(EX\varphi) := \big\{(\vec{q},\vec{x},\vec{Z}) \;\big|\; \forall\vec{q}\,'\exists\vec{x}\,'\forall\vec{Z}\,'\colon \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}\,',\vec{x}\,',\vec{Z}\,')\big) \in R_E \rightarrow (\vec{q}\,',\vec{x}\,',\vec{Z}\,') \in Sat_A(\varphi)\big\}$$

Computation rules for all remaining CTL operators can be build in an analogous manner, see [6] for details.

3 Model Checking for Incomplete Designs with Assumptions

We will now motivate our approach by considering the following two simple CTL formulas for the example given in Fig. 1: $\varphi_1 = AG(\overline{q} \rightarrow EFq)$ and $\varphi_2 = AG(q \rightarrow EF\overline{q})$.

When symbolic model checking for incomplete designs is applied to φ_1, it can be seen that $Sat_A(\varphi_1)$, the set of states surely satisfying φ_1, includes all states in the state space, and thus, our example surely satisfies φ_1.

However, φ_2 cannot be proven so easily, since model checking shows that only the states in which $q = 1$ and $Z = 1$ holds, *surely* satisfy φ_2. On the other side, all states in the state space *possibly* satisfy φ_2 and thus, no result at all can be provided for φ_2.

Now, consider that some additional information about the behavior of the Black Box is given, e.g. that every time that the Black Box input q holds the value 1, in the next state the Black Box output Z holds the value 1: '$q \rightarrow X\,Z$' ('X' stands for 'next'). It can be seen that for all possible replacements of the Black Box for which this *assumption* holds (e.g. a constant 1, a flipflop, etc.), φ_2 will be surely satisfied for the overall design.

The key idea of including additional information about the behavior of the Black Boxes in the model checking run is to restrict the set of possible transitions and states to those that meet the demands of the assumptions.

In this chapter, we will first formally introduce Black Box assumptions and show how to compute a representation of the restrictions the assumptions impose on the Black Boxes' behavior. We will then show how to modify the EX preimage computation in model checking for incomplete designs so that only those possible transitions that are compliant to all the assumptions are considered. Based on this EX preimage computation, it is possible to evaluate any CTL property for the incomplete design, taking the restrictions imposed by the component assumptions into account.

3.1 Black Box Assumptions

Here, we restrict ourselves to assumptions talking about signal values in the current and the next state. However note that it is easy to reduce assumptions talking about states in a fixed time window of future states to the kind of assumptions considered here (e.g. by using auxiliary variables).

Definition 1 *Let V be the set of Black Box in- and outputs. The set of Black Box assumptions is defined by the following BNF:*

$$\begin{aligned}\langle current \rangle &::= v \in V \mid (\langle current \rangle \vee \langle current \rangle) \mid \neg \langle current \rangle \\ \langle currentornext \rangle &::= \langle current \rangle \mid X(\langle current \rangle) \\ \langle assumption \rangle &::= \langle currentornext \rangle \mid (\langle assumption \rangle \vee \langle assumption \rangle) \mid \neg \langle assumption \rangle\end{aligned}$$

We will now show how to recursively compute a relation $A(\psi)$ that includes all pair of states $\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big)$ for which a transition would be compliant to the assumption ψ.

If $\psi = v \in V$ is an input or an output of a Black Box, then all states $(\vec{q},\vec{x},\vec{Z})$ in which v is possibly 1 is put into $A(v)$ (together with arbitrary $(\vec{q}',\vec{x}',\vec{Z}')$):

$$A(v) := \big\{\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \mid (\vec{q},\vec{x},\vec{Z}) \models Sat_E(v)\big\}$$

For $\neg\psi$, $A(\neg\psi)$ includes all compliant pairs of states that are *not* compliant to assumption ψ:

$$A(\neg\psi) := \big(\mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \times \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|}\big) \setminus A(\psi)$$

For $(\psi_1 \vee \psi_2)$, we define $A(\psi_1 \vee \psi_2)$ to include the pairs of states compliant to ψ_1 and the pairs of states compliant to ψ_2:

$$A(\psi_1 \vee \psi_2) := A(\psi_1) \cup A(\psi_2)$$

Finally, $A(X(\psi))$ includes all pairs of states $\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big)$ for which there is a succeeding pair of states $\big((\vec{q}',\vec{x}',\vec{Z}'),(\vec{q}'',\vec{x}'',\vec{Z}'')\in A(\psi)\big)$:

$$A(X(\psi)) := \big\{\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \mid \exists \vec{q}''\, \exists \vec{x}''\, \exists \vec{Z}'' \colon \big((\vec{q}',\vec{x}',\vec{Z}'),(\vec{q}'',\vec{x}'',\vec{Z}'')\big) \in A(\psi)\big\}$$

Other operators like '$\wedge$', '$\rightarrow$' etc. can be expressed by the operators above in the usual way.

Based on $A(\psi)$ for an assumption ψ, the set of possible transitions that are also compliant to ψ can be build by the intersection of R_E and $A(\psi)$.

However, it may happen that for some state (q^1, x^1, Z^1) and for some input value x^2 in the succeeding state with $q^2 = \delta(q^1, x^1, Z^1)$, there is no Black Box output value Z^2 such there is a possible assumption compliant transition from (q^1, x^1, Z^1) to (q^2, x^2, Z^2). In other words, coming from (q^1, x^1, Z^1) and with the current primary input value x^2, there is not a single possible assumption compliant output value of the Black Boxes. We call states with this property not compliant to ψ. Obviously, there is no assumption compliant substitution of the Black Boxes for which state (q^1, x^1, Z^1) is reachable.

Thus, we can also consider all transitions leading to (q^1, x^1, Z^1) as not compliant to the assumptions. Since this may lead to a 'chain reaction' removing even more possible transitions, we use the fixed point iteration given in Fig. 3 to compute the set of assumption compliant transitions $AT(\psi) \subseteq \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|} \times \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|}$. In this fixed point iteration, $AS(\psi) \subseteq \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|}$, the set of states compliant to ψ is also computed.

Example We now compute $A(\psi)$, $AS(\psi)$ and $AT(\psi)$ for our example in Fig. 1 and the assumption $\psi = (q \rightarrow X(Z))$. The recursive construction of $A(q \rightarrow X(Z))$ returns

$$\begin{aligned}A(q \rightarrow X(Z)) = \big\{&\big((\overline{q},\overline{Z}),(\overline{q}',\overline{Z}')\big),\big((\overline{q},\overline{Z}),(\overline{q}',Z')\big),\big((\overline{q},\overline{Z}),(q',\overline{Z}')\big),\big((\overline{q},\overline{Z}),(q',Z')\big),\\ &\big((\overline{q},Z),(\overline{q}',\overline{Z}')\big),\big((\overline{q},Z),(\overline{q}',Z')\big),\big((\overline{q},Z),(q',\overline{Z}')\big),\big((\overline{q},Z),(q',Z')\big),\\ &\big((q,\overline{Z}),(\overline{q}',Z')\big),\big((q,\overline{Z}),(q',Z')\big),\big((q,Z),(\overline{q}',Z')\big),\big((q,Z),(q',Z')\big)\big\}\end{aligned}$$

The fixed point iteration stops after one cycle, since then all states can be seen to be compliant.

$$\begin{aligned}AT(q \to X(Z)) = A(q \to X(Z)) \cap R_E = \big\{&\big((\overline{q},\overline{Z}),(q',\overline{Z}')\big),\big((\overline{q},\overline{Z}),(q',Z')\big),\big((\overline{q},Z),(q',\overline{Z}')\big),\\ &\big((\overline{q},Z),(q',Z')\big),\big((q,\overline{Z}),(q',Z')\big),\big((q,Z),(\overline{q}',Z')\big)\big\}\\ AS(q \to X(Z)) = \big\{&(\overline{q},\overline{Z}),(\overline{q},Z),(q,\overline{Z}),(q',Z')\big\}\end{aligned}$$

The set of transitions for this example is illustrated in Fig. 2; solid arrows illustrate assumption compliant transitions, dashed arrows illustrate transitions, that are no longer considered, since they are not compliant to the assumption.

3.2 Model Checking

The resulting assumption compliant transition relation AT can now be used in analogy to model checking for incomplete designs without assumptions, replacing R_E for the computation of $Sat_A^{\psi}(EX\varphi)$ and $Sat_E^{\psi}(EX\varphi)$ under assumption ψ.

A state $(\vec{q},\vec{x},\vec{Z})$ *possibly* satisfies $EX\varphi$, if there is a possible assumption compliant transition to a state $(\vec{q}',\vec{x}',\vec{Z}')$ possibly satisfying ψ:

$$Sat_E^{\psi}(EX\varphi) := \big\{(\vec{q},\vec{x},\vec{Z}) \mid \exists\vec{q}'\exists\vec{x}'\exists\vec{Z}'\colon \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \in AT(\psi) \wedge (\vec{q}',\vec{x}',\vec{Z}') \in Sat_E^{\psi}(\varphi)\big\}$$

A state $(\vec{q},\vec{x},\vec{Z})$ *surely* satisfies $EX\varphi$, if there is successive input value $\vec{x}'$ such that for all Black Box output values $\vec{Z}'$: If $\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big)$ is an assumption compliant transition, $(\vec{q}',\vec{x}',\vec{Z}')$ surely satisfies ψ:

$$Sat_A^{\psi}(EX\varphi) := \big\{(\vec{q},\vec{x},\vec{Z}) \mid \forall\vec{q}'\exists\vec{x}'\forall\vec{Z}'\colon \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \in AT(\psi) \to (\vec{q}',\vec{x}',\vec{Z}') \in Sat_A^{\psi}(\varphi)\big\}$$

Figure 4 illustrates the computation of $Sat_E^{\psi}(EX\varphi)$ and $Sat_A^{\psi}(EX\varphi)$.

The remaining CTL operators can be adapted without modification from regular model checking for incomplete designs [6].

The final state sets represented by $Sat_A^{\psi}(\varphi)$ and $Sat_E^{\psi}(\varphi)$ can be evaluated as follows: If all *assumption compliant* states $(\vec{q},\vec{x},\vec{Z})$ from the initial state set I are also in $Sat_A^{\psi}(\varphi)$, then φ is valid under the assumption ψ:

$$\forall\vec{q}\forall\vec{x}\forall\vec{Z}\colon \Big(\big((\vec{q},\vec{x},\vec{Z}) \in I \wedge (\vec{q},\vec{x},\vec{Z}) \in AS(\psi)\big) \to (\vec{q},\vec{x},\vec{Z}) \in Sat_A^{\psi}(\varphi)\Big)$$
$$\Longrightarrow \varphi \text{ is valid under assumption } \psi$$

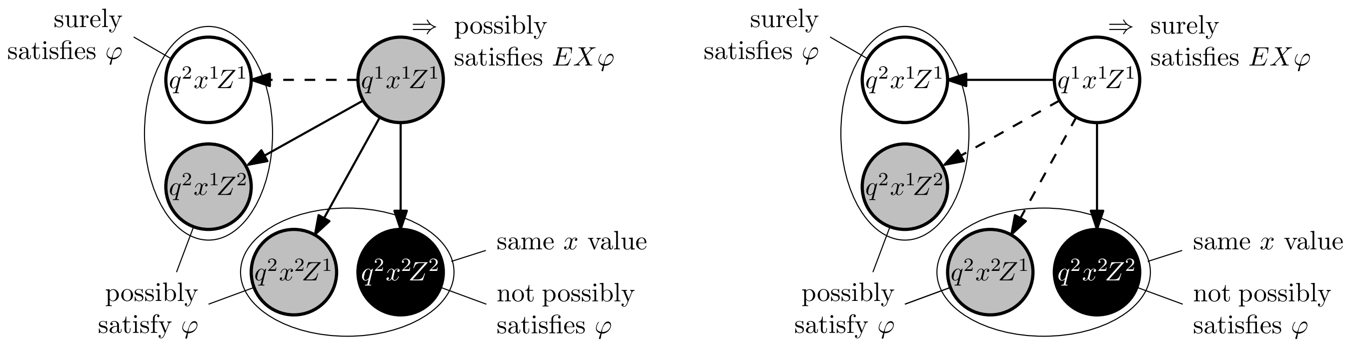
Suppose that there are $\vec{q}$ and $\vec{x}$ such that for all $\vec{Z}$ the following holds: $(\vec{q},\vec{x},\vec{Z})$ is in the initial state set I and if $(\vec{q},\vec{x},\vec{Z})$ is *assumption compliant*, then it is *not* in the set of states possible satisfying φ. Then we can conclude that φ is not realizable under the assumption ψ.

$$\exists\vec{q}\exists\vec{x}\forall\vec{Z}\colon \Big((\vec{q},\vec{x},\vec{Z}) \in I \wedge \big((\vec{q},\vec{x},\vec{Z}) \in AS(\psi) \to (\vec{q},\vec{x},\vec{Z}) \notin Sat_E^{\psi}(\varphi)\big)\Big)$$
$$\Longrightarrow \varphi \text{ is not realizable under assumption } \psi$$

Figure 5 illustrates the evaluation of $Sat_E^{\psi}(\varphi)$ and $Sat_A^{\psi}(\varphi)$.

```
$AT(\psi) := A(\psi) \cap R_E$;
while (!fixed point) {
                                 // compute the set of assumption compliant states $AS(\psi)$
  $AS(\psi) := \big\{(\vec{q},\vec{x},\vec{Z}) \mid \exists\vec{q}'\forall\vec{x}'\exists\vec{Z}'\colon \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \in AT(\psi)\big\}$;
                                 // restrict $AT(\psi)$ to transitions within the set of assumption compliant states
  $AT(\psi) := \big\{\big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \mid \big((\vec{q},\vec{x},\vec{Z}),(\vec{q}',\vec{x}',\vec{Z}')\big) \in AT(\psi) \wedge (\vec{q},\vec{x},\vec{Z}) \in AS(\psi) \wedge (\vec{q}',\vec{x}',\vec{Z}') \in AS(\psi)\big\}$;
}
```

Figure 3: Fixed point computation of $AT(\psi)$ and $AS(\psi)$



surely satisfies φ
$\Rightarrow$ possibly satisfies $EX\varphi$
$q^2x^1Z^1$
$q^1x^1Z^1$
$q^2x^1Z^2$
$q^2x^2Z^1$
$q^2x^2Z^2$
same x value
possibly satisfy φ
not possibly satisfies φ
surely satisfies φ
$\Rightarrow$ surely satisfies $EX\varphi$
$q^2x^1Z^1$
$q^1x^1Z^1$
$q^2x^1Z^2$
$q^2x^2Z^1$
$q^2x^2Z^2$
same x value
possibly satisfy φ
not possibly satisfies φ

Figure 4: Computation of $Sat_E^{\psi}(EX\varphi)$ and $Sat_A^{\psi}(EX\varphi)$. Dashed arrows illustrate transitions not compliant to the assumption.

Example To check the validity of $\varphi_2 = AG(q \rightarrow EF\overline{q})$ for our example under the assumption of $\psi = (q \rightarrow X(Z))$, we first iteratively compute $Sat_A^{\psi}(EF\overline{q})$:

$$\begin{aligned} Sat_A^{\psi}(\overline{q}) &= \{(\overline{q},\overline{Z}),(\overline{q},Z)\} \\ Sat_A^{\psi}(\overline{q} \vee EX\overline{q}) &= \{(\overline{q},\overline{Z}),(\overline{q},Z),(q,Z)\} \\ Sat_A^{\psi}(\overline{q} \vee EX\overline{q} \vee EXEX\overline{q}) &= \{(\overline{q},\overline{Z}),(\overline{q},Z),(q,\overline{Z}),(q,Z)\} \\ \Rightarrow Sat_A^{\psi}(EF\overline{q}) &= \{(\overline{q},\overline{Z}),(\overline{q},Z),(q,\overline{Z}),(q,Z)\} \end{aligned}$$

It is easy to see that $AG(q \rightarrow EF\overline{q})$ is also satisfied for every state in the design and thus, the property is valid under assumption $(q \rightarrow X(Z))$.

3.3 Symbolic Computation

For symbolic computation, we do not simply build a symbolic version of $AT(\psi)$, since this would mean that we have to build a monolithic transition relation which would have a large BDD representation, slowing down the model checking process.

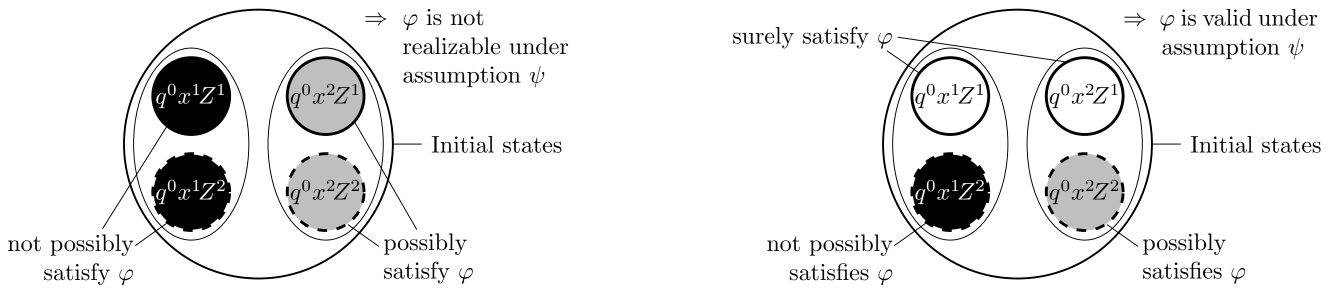
Instead, we just build a symbolic version of $A(\psi)$ and $AS(\psi)$ and perform a functional preimage computation that has been extended to consider both $A(\psi)$ and $AS(\psi)$.

Details are omitted due to page limitations.

4 Assumption Selection

Given an incomplete design and a set of assumptions $\Psi = \{\psi_1, \ldots, \psi_n\}$ for the Black Boxes in the design, it is possible to perform model checking under consideration of all assumptions by building an assumption $\psi = \bigwedge_{1 \leq i \leq n} \psi_i$ and invoking the model checking routine as described above.

However, considering *all* assumptions ψ_i may be unnecessary, especially when the set of assumptions provide an exhaustive description of the Black Boxes' behavior, which would be equivalent to model checking of a complete design. Thus, our goal is to find a small set of assumptions that is sufficient to prove the unrealizability or validity of the given property, but at the same time does not need the computational resources that the consideration of all assumptions would require.



$\Rightarrow$ φ is not realizable under assumption ψ
$q^0x^1Z^1$
$q^0x^2Z^1$
$q^0x^1Z^2$
$q^0x^2Z^2$
Initial states
not possibly satisfy φ
possibly satisfy φ
surely satisfy φ
$\Rightarrow$ φ is valid under assumption ψ
$q^0x^1Z^1$
$q^0x^2Z^1$
$q^0x^1Z^2$
$q^0x^2Z^2$
Initial states
not possibly satisfies φ
possibly satisfies φ

Figure 5: Evaluation of $Sat_E^{\psi}(\varphi)$, $Sat_A^{\psi}(\varphi)$. States not compliant to the assumption have dashed outlines.

4.1 ACTL

For the time being, we will restrict ourselves to invariants of the form $AG\gamma$ where γ is a property without any temporal operators.

We now describe a heuristic method that, starting from no assumption at all, iteratively takes more and more assumptions into consideration until the model checking routine is able to prove either the validity or the unrealizability.

This approach follows the idea of Counterexample Guided Abstraction Refinement (CEGAR) [7], in which an initial abstraction is refined by means of analyzing counterexamples.

Here, we compute 'possible counterexamples', sequences of states that are connected by possible transitions compliant to the current set of assumptions, and in which the last state does not surely satisfy γ. Such a possible counterexample exists iff the validity of the property $AG\gamma$ cannot be proven. This possible counterexample can then be compared against the remaining assumptions that were not yet considered; if there is an assumption that is not satisfied for a transition in the possible counterexample, this assumption can be added to the set of considered assumptions, henceforth preventing the current possible counterexample.

Let $\mathcal{A} \subseteq \Psi$ be the set of assumptions that are currently considered and $\mathcal{B} = \Psi \setminus \mathcal{A}$ be the set of available assumptions not considered so far. The set of possible transitions compliant to all assumptions in $AT(\mathcal{A})$ is build by $AT(\mathcal{A}) := AT\big(\bigwedge_{\psi\in\mathcal{A}} \psi\big)$.

A possible counterexample of length n to a CTL formula $AG\gamma$ under consideration of assumptions $\mathcal{A}$ is a sequence of states $(s_i) = (s_0, s_1, ..., s_{n-1})$ with $s_i = (\vec{q_i}, \vec{x_i}, \vec{Z_i})$ where s_0 is an initial state, there is a possible transition from s_i to s_{i+1} for all $i \in \{0, ..., n-2\}$ that is compliant to all assumptions in $\mathcal{A}$ and the last state s_{n-1} does not surely satisfy γ.

The possible image and possible preimage computation of some state set $X \subseteq \mathbb{B}^{|\vec{q}|} \times \mathbb{B}^{|\vec{x}|} \times \mathbb{B}^{|\vec{Z}|}$ under assumptions $\mathcal{A}$ can be build in analogy to the computation of $Sat_E(EX\varphi)$:

$$\text{preimg}^{\mathcal{A}}(X) := \big\{(\vec{q}, \vec{x}, \vec{Z}) \mid \exists\vec{q}'\exists\vec{x}'\exists\vec{Z}'\colon \big((\vec{q}, \vec{x}, \vec{Z}), (\vec{q}', \vec{x}', \vec{Z}')\big) \in AT(\mathcal{A}) \wedge (\vec{q}', \vec{x}', \vec{Z}') \in X\big\}$$
$$\text{img}^{\mathcal{A}}(X) := \big\{(\vec{q}', \vec{x}', \vec{Z}') \mid \exists\vec{q}\exists\vec{x}\exists\vec{Z}\colon \big((\vec{q}, \vec{x}, \vec{Z}), (\vec{q}', \vec{x}', \vec{Z}')\big) \in AT(\mathcal{A}) \wedge (\vec{q}, \vec{x}, \vec{Z}) \in X\big\}$$

Based on this definition of possible image and possible preimage computation, a possible counterexample can be computed by using the algorithm given in Fig. 6. The algorithm can be extended to compute possible counterexamples for all ACTL formulas for which a linear counterexample can be computed [12].

The received possible counterexample can be compared against the not yet considered assumptions. If a transition $((q_i, x_i, Z_i), (q_i', x_i', Z_i'))$ in the possible counterexample is not compliant to some assumption $\psi \in \mathcal{B}$ ('ψ *breaks* transition $((q_i, x_i, Z_i), (q_i', x_i', Z_i'))$'), we can include this assumption into $\mathcal{A}$ and by that forbid the behavior the Black Boxes showed in the possible counterexample, in the next model checking run.

We considered different heuristics to select the assumptions that will be considered in the next model checking run. Our first heuristic simply adds the first assumption in $\mathcal{B}$ that can be shown to break some transition in the possible counterexample. This can be improved by first counting how often each assumption breaks the current possible counterexample; of those assumptions that break the possible counterexample most often, we then either pick a single one or add all those assumptions at once.

```
$Q_0 := Sat_E(\neg\gamma) \cap AS(\mathcal{A})$        // set of states not surely satisfying $\gamma$.
$i := 0$
*while*$(I \cap Q_i = \varnothing)$ {                  // iterate until intersection with initial state set $I$
    $Q_{i+1} :=$ preimg$^{\mathcal{A}}(Q_i)$            // compute preimage of previous $Q$-set
    $i := i + 1$
}
$s_0 :=$ choose-state$(I \cap Q_i)$          // choose an initial state that is also in the last $Q$ state set.
$n := i + 1$;
*while*$(i > 0)$ {
    $s_{n-i} :=$ choose-state(img$^{\mathcal{A}}(\{s_{n-i-1}\}) \cap Q_{i-1}$) // choose a state from the intersection
                                             // of the image of the last state and the next $Q$ state set.
    $i := i - 1$;
}
```

Figure 6: Possible counterexample computation

4.2 Full CTL

Computing linear possible counterexamples is only possible for a subset of ACTL [12]. We now briefly sketch two additional heuristics applicable for full CTL.

1. *Evaluating preimage computation:* In the recursive evaluation of the CTL formula, the model checking routine frequently computes the (possible or sure) preimage of some set of states S. For each preimage computation, we look at the impact each additional assumption $\psi \in \mathcal{B}$ would have on the result. More specific, we compute the cardinality of the set of possible transitions compliant to the assumptions $\mathcal{A} \cup \{\psi\}$, leading from the preimage of S to S itself for every $\psi \in \mathcal{B}$. The intuition behind this is that less possible transitions will probably lead to a more exact result. Thus, the $\psi \in \mathcal{B}$ that would have led to the largest reduction of the number of possible transitions in the current iteration, is added to $\mathcal{A}$ in the next iteration.

2. *Cardinality of AT:* $AT(\psi)$ contains all possible transitions of the system which are compliant to assumption ψ. The cardinality of $AT(\psi)$ can be seen as a measure for the impact of assumption ψ to the system. The less possible transitions $AT(\psi)$ contains, the more the behavior of the Black Boxes is potentially constrained by ψ. Thus, we compare the cardinalities of $AT(\psi)$ for every $\psi \in \mathcal{B}$ and include the assumption(s) in descending order of these values.

5 Case Study

To evaluate our method we used the FFB benchmark suite [13]. The FFB benchmark suite is based on the rail segment control (Fahrwegesteuerung) from the FunkFahrBetrieb (FFB) specification of the Deutsche Bahn [14], which is closely related to the European Train Control System level 2/3 Movement Authority [15]. A FFB benchmark models a network of segments and trains that move on these segments according to a given schedule.

The main concept of this benchmark is as follows: Each train sends requests to the current and subsequent segments (determined by the train's schedule). Some time later, the segments will grant permission to the train by sending them an acknowledge signal. A train only moves on segments it has permission to. If a train has moved from a segment, it no longer sends a request.

In our experiments we modeled all segments in the design as Black Boxes. For each of these Black Boxes, we specified a set of assumptions (n is the number of trains, m is the number of segments; $Req_{i,k}$ denotes a request signal from train i to segment k and $Ack_{k,i}$ denotes an acknowledge signal from segment k to train i). For each segment k:

1. At every point in time, the segment sends at most one acknowledge signal Ack to a train $i \in \{1, \dots, n\}$: $\left(Ack_{i,k} \rightarrow \bigwedge_{j=1,\, j\neq i}^{n} \neg Ack_{j,k}\right)$

2. An acknowledge signal to a train $i \in \{1, \dots, n\}$ is held if there still is an according request signal: $\left(Ack_{k,i} \wedge Req_{i,k}\right) \rightarrow X Ack_{k,i}$

3. If there is at least one request in the current state, there will be at least one acknowledge signal in the next state: $\left(\bigvee_{i=1}^{n} Req_{i,k}\right) \rightarrow X\left(\bigvee_{i=1}^{n} Ack_{k,i}\right)$

4. A train only will receive an acknowledge signal in the next state, if it is sending a request in the current state: $\bigwedge_{i=0}^{n-1} \left(X Ack_{k,i} \rightarrow Req_{i,k}\right)$

Note that these assumptions are not an exhaustive description of the segments; for instance, it is not specified, which of two trains simultaneously requesting permission receives the first acknowledge signal when the segment is currently free (in the complete design, a round-robin arbiter is used for this matter).

For our experiments we used a FFB design with 4 trains on 16 segments; all segments were put into Black Boxes. For each segment we defined 14 assumptions as explained above, resulting in a total number of 224 assumptions. We present the results for three invariants φ_1, φ_2 and φ_3, each stating that a specific pair of trains is never on the same segment at the same time, i.e. these two trains never collide. Furthermore we give results for two non-ACTL properties φ_4, φ_5, stating that a certain train always has the possibility to move on at some time in the future.

For each of the properties, we compared a model checking run for the complete design without Black Boxes (‘complete’) to a run for the incomplete design that considered all assumptions (‘all’).

Additionally, we evaluated the incremental approach that starts with no assumption and then heuristically adds assumptions until the property can be shown to be valid or unrealizable. For ACTL formulas, we applied three different counterexample-based heuristics as described in Sect. 4.1: Add the first assumption that can be shown to break the current possible counterexample (‘first’); add a single assumption that breaks the current possible counterexample most often (‘single’); add the set of assumptions that break the current possible counterexample most often (‘multiple’).

Finally, we evaluated two heuristics that can be applied for all CTL properties (as described in Sect. 4.2): Add all assumptions with the greatest impact on preimage computation (‘preimage’); add the assumptions with the smallest cardinality of the according AT relation (‘AT’).

For each set of experiments, the tables give the total time in CPU seconds (‘time’), the symbolic model checking time (‘SMC’), the time used for assumption selection (‘select’), the maximum number of considered assumptions (‘#as’), the number of iterations (‘#iter’) and finally the peak number of BDD nodes (‘#nodes’).

Our prototype model checker that is based on the BDD package CUDD 2.4.1 [16] implements symbolic model checking for incomplete designs with assumptions as well as the assumptions selection methods. All state sets and assumption relations are represented symbolically. All experiments were performed on a AMD Opteron processor running on 2.6 GHz and with 4 GB of main memory. We used a timeout of 7200 CPU seconds.

The preliminary results for our experiments are given in Tab. 1. As can be seen, the run times for symbolic model checking considering all assumptions are often higher than for the complete design, validating our approach to incrementally select a subset of assumptions that are considered.

In the cases that a possible counterexample could be computed (φ_1, φ_2, φ_3), the heuristic method that picks multiple assumptions by this counterexample performs best. It clearly outperforms the approach of adding all component assumptions at once and also outperforms the complete case in two of the benchmarks. For the single benchmark the heuristic performed worse, the increase in computational time was less than 70%, while for the others, it performed 4 to 6 times faster than the complete design.

The heuristics that are not based on counterexample evaluation on the other hand included too many assumptions that were not necessary for proving the property and thus did seldom perform better than considering all assumptions at once. Here we definitely see room for improvement of the heuristics.

For properties φ_1, φ_3 and φ_4, the approach abstracting the complete design by component assumptions clearly pays off in comparison to model checking for the complete design (which even has a timeout for property φ_4).

Based on experiences made with model checking for incomplete designs we expect that the variants using component assumptions only for a subset of Black Boxes can be improved even further: Our results

φ_1	time	SMC	select	#as	#iter	#nodes
complete	130.70					621798
all	2045.50			224		1652544
first	5849.99	1954.87	3878.81	81	81	2921452
single	28.13	5.45	7.84	42	42	19238
multiple	21.59	3.84	3.19	46	14	24756
preimage	4889.65	2130.35	2731.46	206	29	1595630
AT	476.19	407.42	0	208	4	746725

φ_2	time	SMC	select	#as	#iter	#nodes
complete	242.32					503772
all	timeout					
first	timeout					
single	timeout					
multiple	411.20	281.84	114.51	64	25	446492
preimage	timeout					
AT	775.40	704.99	0	208	4	1629540

φ_3	time	SMC	select	#as	#iter	#nodes
complete	108.87					344911
all	1777.48			244		2249814
first	6905.91	3831.74	3057.57	78	78	2537200
single	55.78	19.94	19.91	58	58	83478
multiple	25.12	4.58	5.19	55	23	22818
preimage	657.41	14.58	619.54	148	37	650687
AT	299.03	259.29	0	208	4	766362

φ_4	time	SMC	select	#as	#iter	#nodes
complete	timeout					
all	1308.23			224		2216450
preimage	timeout					
AT	1443.08	1402.48	0	224	5	1740366

φ_5	time	SMC	select	#as	#iter	#nodes
complete	23.66					194663
all	3020.56			224		1989280
preimage	timeout					
AT	3085.95	3045.55	0	224	5	2040170

Table 1: Experimental results

in part suffer from the fact that in our preliminary implementation the outputs of all Black Boxes are always modeled by symbolic Z_i-simulation, i.e., there is a variable for each Black Box output in the state space during symbolic model checking. For model checking for incomplete designs, we made the experience that run times can be tremendously improved when Black Box outputs are modeled using the so-called symbolic $(0, 1, X)$-simulation [10], that abstracts Black Box outputs by an unknown value X, thus removing many variables Z_i from the state space. We expect that for Black Box outputs that are not yet restricted by component assumptions this abstraction will lead to considerable performance gains in the context of assumption based model checking as well.

6 Conclusion and Future Work

We introduced a new symbolic model checking method that is able to consider local component properties ('assumptions') for the Black Boxes in an incomplete design. Our method is able to process full CTL and to provide proofs of validity (the property is satisfied for all replacements of the Black Boxes that are compliant to the local component properties) and unrealizability (the property is not satisfied for any replacement of the Black Boxes that is compliant to the local component properties).

We provided heuristics to iteratively select those assumptions that are necessary to provide a proof on a CEGAR basis; we also gave some ideas how to select assumptions for arbitrary CTL formulas for which no possible counterexample can be computed.

Currently we are working on an enhancement on the model checking routine that automatically uses a less exact but more efficient modeling like symbolic $(0, 1, X)$-modeling for Black Box outputs that are not (yet) restricted by component assumptions.

References

[1] A. P. Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[2] Edmund M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[3] J. R. Burch, Edmund M. Clarke, Kenneth L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^20 States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

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

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

[6] Tobias Nopper and Christoph Scholl. Approximate symbolic model checking for incomplete designs. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design*, volume 3312 of *LNCS*, pages 290–305, Austin, Texas, November 2004. Springer Verlag.

[7] Edmund M. Clarke, Orna Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *Int'l Conf. on CAV*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[8] C. Braunstein and E. Encrenaz. Using ctl formulae as component abstraction in a design and verification flow. *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*, pages 80–89, July 2007.

[9] Martin Schickel, Volker Nimbler, Martin Braun, and Hans Eveking. *An Efficient Synthesis Method for Propery-Based Design in Formal Verification*, chapter 10. Kluver, 2007.

[10] Christoph Scholl and Bernd Becker. Checking equivalence for partial implementations. In *ACM IEEE Design Automation Conf.*, pages 238–243, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[11] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[12] F. Buccafurri, T. Eiter, G. Gottlob, and N. Leone. On ACTL formulas having linear counterexamples. *Journal of Computer and System Sciences*, 62(3):463–515, 2001.

[13] AVAVS S1. The FFB Benchmarks. http://www.avacs.org/, 2007.

[14] ADtranz Deutsche Bahn AG. *Betriebliches Lastenheft FunkFahrBetrieb (FFB) 413.9210*, version 2.0, 01.10.1996 edition, 1997.

[15] ERTMS/ETCS - Class 1 — system requirements specification, chapter 3 (principles), subset-026-3, issue 2.2.2. 2002.

[16] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 2.4.0*. University of Colorado at Boulder, 2004.