Preprint from Proceedings of International Conference on Computer Aided Design (ICCAD 2007), November 2007, San Jose, USA

Computation of Minimal Counterexamples by Using Black Box Techniques and Symbolic Methods

Tobias Nopper and Christoph Scholl and Bernd Becker
Institute of Computer Science, Albert-Ludwigs-University, D-79110 Freiburg, Germany
{nopper, scholl, becker}@informatik.uni-freiburg.de

***Abstract*— Computing counterexamples is a crucial task for error diagnosis and debugging of sequential systems. If an implementation does not fulfill its specification, counterexamples are used to explain the error effect to the designer. In order to be understood by the designer, counterexamples should be simple, i.e. they should be as general as possible and assign values to a minimal number of input signals.**

Here we use the concept of *Black Boxes* — parts of the design with unknown behavior — to mask out components for counterexample computation. By doing so, the resulting counterexample will argue about a reduced number of components in the system to facilitate the task of understanding and correcting the error. We introduce the notion of 'uniform counterexamples' to provide an exact formalization of simplified counterexamples arguing only about components which were not masked out. Our computation of counterexamples is based on symbolic methods using AIGs (And-Inverter-Graphs). Experimental results using a VLIW processor as a case study clearly demonstrate our capability of providing simplified counterexamples.

I. Introduction

Given a sequential circuit and properties in some temporal logic like CTL or LTL, model checking is a method for verifying these properties [1], [2]. In the early nineties, by introducing *symbolic* model checking, Burch et al. substantially extended the class of systems which can be verified [3], [4]. In (unbounded) symbolic model checking, Binary Decision Diagrams (BDDs) [5] (or more recently And-Inverter Graphs (AIGs) [6], [7], [8]) are used both for state set representation and for state traversal. In the last few years SAT based techniques like Bounded Model Checking (BMC) [9], [10] have also attracted much interest. BMC applied to certain properties (safety properties or, more generally, LTL formulas) 'unfolds' the transition relation for k steps and checks whether there is a run of length k which does not fulfill the specified property. If such a run does not exist, then k is increased and BMC is used again. However, for *proving* properties using BMC, a suitable upper bound on k is needed. In the case of safety properties, e.g., the procedure can be stopped, when k equals the *diameter* of the system, i.e. the maximum length of all shortest paths between states in the system. In this case, BMC ends up with a proof of the property. Since computing the diameter of a system turns out to be difficult however, alternative approaches such as k-induction [11] and interpolation [12] have been proposed for making SAT-based model checking complete.

In this paper we address an important feature of *unbounded symbolic* model checkers, the generation of counterexamples for cases when the specified property does not hold. If the property does not hold, counterexamples are an important means for the designer, since they explain the error by showing a run of the system violating the property and may guide the designer to a correction of the error in the system. In this sense the strength of model checking as a verification technique largely relies on counterexamples produced for failing properties.

Standard methods for counterexample generation implemented in model checkers like SMV [3], [4] produce a sequence of assignments to all input signals defining an erroneous run of the system. Thus, a consideration of all parts of the system is needed in order to understand and reproduce the error effect in the current design. In this paper we consider a method for constructing counterexamples arguing about a reduced number of components of the system in order to facilitate the task of understanding and correcting the error.

The goals of our work are most closely related to approaches in the Bounded Model Checking (BMC) context which try to improve a counterexample produced by a SAT solver in BMC [13], [14]. Starting from a single counterexample which specifies all signals at all times prior to the error, the approach from [13] tries to remove ('lift') assignments to input signals without losing the property that the remaining assignments imply a violation of the property. It is important to note that the quality of this method strongly depends (1) on the order which is used for removing assignments to input signals and (2) on the original counterexample produced by the SAT solver (which may happen to be only one out of a large number of possible counterexamples).[1]

In contrast, our approach to the improvement of counterexamples in symbolic model checking is based on Black Box model checking methods [15] where components of a system are removed and replaced by so-called Black Boxes. If Black Box model checking is able to prove that the property is violated *independently from the implementation of the Black Boxes*, then we know that those parts of the design which were replaced by Black Boxes are not important for explaining the error effect and we have successfully reduced the number of components to be considered when analyzing the error. In our current approach Black Boxing is performed interactively based on the property and conjectures of the user about which parts of the design could potentially be irrelevant for the error. Black Boxing may be performed step by step until further replacements of components by Black Boxes would lead to the situation that Black Box model checking is not able to prove a violation of the property independently from the Black Box

[1] We refer to Section III-G for a more detailed discussion comparing SAT based methods to our approach.

1-4244-1382-6/07/$25.00 ©2007 IEEE

implementation.

Another application scenario for our counterexample generation methods presented in this paper are partial designs with Black Boxes where Black Boxes represent parts of the design which are not yet present in early phases of the design process. In order to help the designer to remove errors which are already present in the current partial design, counterexamples are needed which do not depend on values produced by some future implementation of the Black Boxes. However in this paper we focus our attention on the generation of improved counterexamples by removing components from an already existing design.

Based on existing methods for Black Box model checking [15], we had to develop methods for counterexample generation in this context. In Section III we show that a straightforward generalization of counterexample generation from classical symbolic model checking to Black Box model checking will not meet our demands. The reason for this lies in the fact that counterexamples generated by the straightforward method may still argue about the signals at the interface of the Black Boxes and different implementations of the Black Boxes may lead to different counterexamples. Hence we had to develop more sophisticated methods which produce 'uniform' counterexamples that do not depend on the behavior implemented by the Black Boxes.

Note that in contrast to previous approaches dealing with unknown initial states [16], [17], we have to cope with different possible behaviors of the system due to different Black Box implementations since the 'uniform counterexample' is defined to be a counterexample *for all* possible Black Box implementations.

A. Related Work

Apart from approaches in the Bounded Model Checking (BMC) context trying to improve concrete counterexamples by 'lifting' assignments produced by a SAT solver [13], [14] as already mentioned above, there are related methods based on ternary $(0, 1, X)$-logic such as Symbolic Trajectory Evaluation (STE) [18], [19]. The most popular applications of STE are also based on properties arguing about bounded time windows. These properties (called 'simple assertions' in [18]) have the special form $A \Rightarrow C$ where A and C are so-called trajectory formulas. The antecedent A expresses constraints on signals at different times t, and the consequent C expresses requirements that should hold on signals at (some other) times t'. STE solves the model checking problem by considering symbolic representations for all runs of the system ('trajectories') fulfilling A and all sequences of signals fulfilling C. The traces fulfilling A are overapproximated using ternary $(0, 1, X)$-logic. (In contrast, approximations used in our method are not necessarily bound to $(0, 1, X)$-logic, see also Section II.). Counterexamples for failing properties specify only input signals which were mentioned in A or C, other inputs retain the unknown (and thus arbitrary) value X.

However, it may be the case that $A \Rightarrow C$ can not be proved due to overapproximations of traces allowed by A. Then in a refinement step more symbolic variables (or more concrete values) may be introduced into A (reducing unknown values X and thus reducing the set of traces which are represented symbolically). However, introducing symbolic variables for inputs may potentially introduce complex parts of the system into counterexamples which could be 'black boxed' by our method. Introducing symbolic variables for *internal* signals (other than inputs) can lead to the problem of 'vacuously failing' properties [20], i.e. to the problem that there is no concrete counterexample which is compatible to the abstract counterexample derived from A and C by STE.

More general properties for STE arguing about unbounded time windows (like iterations from [18]) suffer from the fact that sets of states have to be approximated by $(0, 1, X)$ (e.g. a state set $\{(00), (01), (10)\}$ can only be described as (XX) [18]). This can have the effect that for faulty designs it is unknown whether a property is violated or not (due to values X in the state vector). In contrast, our method does not approximate sets of states using X-values, even if $(0, 1, X)$-logic is used for approximating transitions in the system.

Another interesting approach which is related to our method by giving more information to the user than a single counterexample was presented by Copty et al. [21]: In their approach most insight into the nature of counterexamples has been given given by so-called 'strong values': Starting from a specific counterexample Copty et al. are able to show, e.g., that some state bits at some instants in time have the same values *for all possible* counterexamples, i.e., these 'strong values' are essential for the error trace. In contrast, our focus is on providing information that certain parts of the circuit and certain signals in the counterexample are *definitely irrelevant* for the existence of the error. The approaches are on the one hand related in the sense that they focus the attention of the user to the most relevant parts in the counterexample, but on the other hand they form two orthogonal methods.

The paper is structured as follows: Section II reviews the main ideas of Black Box model checking which is a necessary ingredient for our counterexample optimization. Then, Sect. III introduces our approach generating 'uniform', minimized counterexamples and presents some interesting theoretical results wrt. the length of uniform counterexamples. Experimental results demonstrating counterexample minimization for a VLIW processor design are presented in Sect. IV. Our experiments rely on symbolic methods using AIGs (And-Inverter-Graphs) and they clearly illustrate that our methods are able to go beyond 'lifting' assignments to input signals in fixed counterexamples produced by SAT solvers. Finally, Sect. V summarizes the paper and gives directions for further research.

II. Preliminaries

In this section we give a brief introduction into symbolic representations of incomplete designs by means of a small sequential example given in Fig. 1. The unknown part of the design has been combined into a so-called 'Black Box'.

For the remainder of this paper, we abbreviate the state names for this example as follows: We use $\overline{q}_1\overline{q}_0$ for $(q_1 = 0, q_0 = 0)$, $\overline{q}_1 q_0$ for $(q_1 = 0, q_0 = 1)$, $q_1\overline{q}_0$ for $(q_1 = 1, q_0 = 0)$, and $q_1 q_0$ for $(q_1 = 1, q_0 = 1)$. Likewise, we use $\overline{x}$ for $x = 0$ and x for $x = 1$.

The Black Box computes a boolean-valued output Z_0. Symbolic simulation of this circuit returns the following result

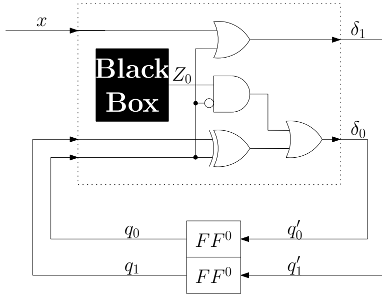

Fig. 1. An exemplary sequential circuit with one Black Box. Initially, $q_1 = q_0 = 0$.

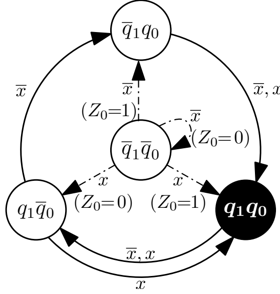

Fig. 2. Automaton of the circuit in Fig. 1 with fixed transitions (solid arrows) and possible transitions (both solid and dashed arrows). $\overline{q}_1\overline{q}_0$ is the initial state, the safety property $AG(\neg q_0 \vee \neg q_1)$ is violated in state $q_1 q_0$.

for the transition functions δ_0 and δ_1:

$$\delta_0(q_1, q_0, x, Z_0) = \overline{q}_0 \cdot Z_0 + \overline{q}_0 \cdot q_1 + q_0 \cdot \overline{q}_1$$
$$\delta_1(q_1, q_0, x, Z_0) = x + q_0$$

Given a state (q_1, q_0) and an input x, the next state computed by the transition functions may or may not depend on the behavior of the Black Box: For instance, for state $\overline{q}_1 q_0$ and input x the next state will be $q_1 q_0$ independently of the value Z_0 computed by the Black Box. In this case we call the transition $(\overline{q}_1 q_0, x, q_1 q_0)$ a *fixed transition*, since it will be present for all possible replacements of the Black Box. On the other hand, for state $\overline{q}_1\overline{q}_0$ and input x the next state may be either $q_1\overline{q}_0$ (in case that the Black Box computes $Z_0 = 0$ in this situation) or $q_1 q_0$ (in case that the Black Box computes $Z_0 = 1$ in this situation). Here we say that both transitions $(\overline{q}_1\overline{q}_0, x, q_1\overline{q}_0)$ and $(\overline{q}_1\overline{q}_0, x, q_1 q_0)$ are *possible transitions*, since they may or may not be present depending on the implementation of the Black Box.

For the complete set R_A of fixed transition we have the following result

$$R_A = \{(\overline{q}_1 q_0, \overline{x}, q_1 q_0), (\overline{q}_1 q_0, x, q_1 q_0), (q_1\overline{q}_0, \overline{x}, \overline{q}_1 q_0), (q_1\overline{q}_0, x, q_1 q_0), (q_1 q_0, \overline{x}, q_1\overline{q}_0), (q_1 q_0, x, q_1\overline{q}_0)\}$$

and for the set R_E of possible transitions we have

$$R_E = \{(\overline{q}_1\overline{q}_0, \overline{x}, \overline{q}_1\overline{q}_0), (\overline{q}_1\overline{q}_0, \overline{x}, \overline{q}_1 q_0), (\overline{q}_1\overline{q}_0, x, q_1\overline{q}_0), (\overline{q}_1\overline{q}_0, x, q_1 q_0), (\overline{q}_1 q_0, \overline{x}, q_1 q_0), (\overline{q}_1 q_0, x, q_1 q_0), (q_1\overline{q}_0, \overline{x}, \overline{q}_1 q_0), (q_1\overline{q}_0, x, q_1 q_0), (q_1 q_0, \overline{x}, q_1\overline{q}_0), (q_1 q_0, x, q_1\overline{q}_0)\}.$$

Figure 2 illustrates the fixed transitions (solid arrows) and the possible transitions (both dashed and solid arrows) for this example.

For understanding the following sections the existence of fixed and possible transitions is the most crucial point.

For the general case, the characteristic functions χ_{R_A} and χ_{R_E} representing the sets R_A resp. R_E may be computed using symbolic methods [15], [22].

In [15], [22], different possibilities to model the behavior of Black Boxes in their environment are considered. Starting from a symbolic version of ternary $(0, 1, X)$-simulation [23] where Black Box outputs are handled by assigning the unknown value X to them, Black Box outputs can alternatively be modeled by distinct Z_i-variables (as already shown above for the small example of Fig. 1). Modeling with distinct Z_i-variables leads to a more accurate representation in general, however at the expense of more variables and higher costs for symbolic representations and manipulations. In a second step, these Z_i variables can also be included into the state space during symbolic model checking, which again results in a gain of accuracy.

The different methods to handle the Black Box outputs lead to different approximations of the sets of *fixed* and *possible* transitions. Based on this, [15] defines a symbolic model checking procedure (for arbitrary CTL formulas) that is able to falsify so-called realizability problems and to prove validity problems for designs with Black Boxes. A property is called realizable, if there is a replacement of the Black Boxes by some implementation such that the overall design fulfills the property and a property is called valid, if it holds independently of the replacement of the Black Boxes. In this paper we restrict our considerations to disproofs of the realizability of safety properties.

III. Counterexample Generation for Incomplete Designs

In this section, we present our method to compute counterexamples for a given incomplete design and a safety property, i.e., a property that has to be satisfied in every state reachable from the initial state.

In the case that symbolic model checking for an incomplete design returns the result that a safety property is not realizable, it is proven that for every Black Box substitution, there is an input sequence leading to a state violating the safety property (hence a counterexample). However it needs to be defined how counterexamples should be computed in this situation.

Since our motivation for Black Boxing was the *simplification* of counterexamples, our goal is to find counterexamples *that work irrespective of a concrete substitution of the Black Boxes*. We now discuss three possible approaches to this.

A. Approaches

a) Straightforward approach: The straightforward approach to generating counterexamples for incomplete designs would consist in an adaption of the standard method for computing counterexamples [24], meaning that the Black Box outputs could — like inputs — hold arbitrary values in every state. Yet, this allows the counterexample computation to make assumptions about the Black Box output values. The

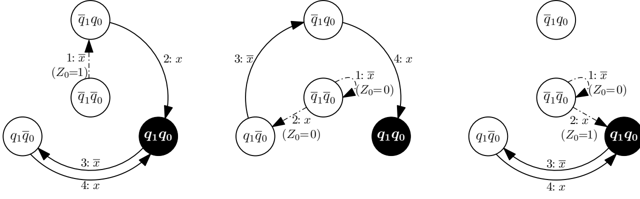

Fig. 3. Illustration of the three possible paths that the system in Fig. 2 can take for the input sequence $\tilde{x} = (0, 1, 0, 1)$ depending on the Black Box substitution. Since all possible paths end in an errorneous state, $\tilde{x}$ is a *uniform* counterexample that works for all replacements of the Black Box.

problem with this approach is illustrated by means of the example in Fig. 2: Suppose the initial state is $\overline{q}_1\overline{q}_0$ and our safety property is $AG(\neg q_0 \vee \neg q_1)$, i.e. the erroneous state is q_1q_0: Then the input sequence of length 1 $\tilde{x} = (1)$ is a counterexample for all Black Box replacements that produce output $Z_0 = 1$ when the sequential circuit is in the initial state with $x = 1$. However, this counterexample does not work for other replacements, e.g. for replacements that produce output $Z_0 = 0$ in this situation, since then the sequence $\tilde{x} = (1)$ would lead us to state $q_1\overline{q}_0$ which is not erroneous. Thus, this approach does not meet our goal to find a counterexample that does not depend on the behavior of the Black Box.

b) Approach based on fixed transitions: Alternatively, it is possible to consider *fixed* transitions in R_A only. By that, the input sequence forces the design into a specific sequence of states, independently from the Black Box substitutions. Thus, if such an input sequence is found, it will work for every Black Box replacement. Yet, in many cases, it is not possible to find such a counterexample, since the erroneous states are not reachable from the initial state by fixed transitions only. For our example in Fig. 2, there is no fixed successor for the initial state $\overline{q}_0\overline{q}_1$ (see R_A for this example as given in Sect. II), and thus no such counterexample can be generated only based on fixed transitions here.

c) Uniform counterexamples: The following approach is not limited to fixed transitions, but still meets our goal of finding counterexamples that work irrespective of substitutions of the Black Boxes. This approach is based on the notions of 'possible paths' and 'uniform counterexamples':

Definition 1: Given an input sequence $(x^1, \ldots, x^n)$ and a state q^0, a *possible path* starting with q^0 and resulting from sequence $(x^1, \ldots, x^n)$ is any sequence of states $(q^0, \ldots, q^n)$ where (q^{i-1}, x^i, q^i) is in the set of possible transitions R_E $(1 \leq i \leq n)$.

Definition 2: A *uniform* counterexample for an incomplete design is a sequence of input values such that each resulting possible path starting with a specific initial state ends with an erroneous state violating the safety property.

Thus, a uniform counterexample works independently of the Black Boxes substitutions. (However, both the intermediate states and the final erroneous state may vary depending on the substitutions of the Black Boxes.)

For our example in Fig. 2, the input sequence $\tilde{x} = (0, 1, 0, 1)$ is a *uniform* counterexample that works for all replacements of the Black Box, since it works for all three possible paths as illustrated in Fig. 3.

B. Computation of Uniform Counterexamples

The main idea of uniform counterexample computation is as follows: Starting from the set of states directly violating the safety property, we iteratively compute all states q with the property that there is an input sequence that leads us from q into an erroneous state, regardless of which possible path is taken. These input sequences are built in the iteration process and are stored together with the states q.

To simplify matters, we will first illustrate our method in an explicit way and show how to perform symbolic computation later. Let Q be the set of states, I be the set of initial states, and X be the set of possible input values. In our case, $Q = \mathbb{B}^{|\vec{q}|}$ and $X = \mathbb{B}^{|\vec{x}|}$.

The safety property φ has the form $\varphi = AG(\psi)$, whereas ψ only consists of state variables, negation, conjunction and disjunction.[2] The set of erroneous states is defined as $\{q \in Q \,|\, q \not\models \psi\}$.

In our algorithm we inductively define sets $C_i \subseteq Q \times X^i$, i.e., an element $(q, \tilde{x}) \in C_i$ consists of a state q and an input sequence $\tilde{x} = (x^1, \ldots, x^i)$ of length i. To prove the correctness of our approach we use the following invariant for every set C_i constructed during our iteration:

$(q, \tilde{x}) \in C_i$, if and only if each possible path starting with q and resulting from input sequence $\tilde{x}$ will end in an erroneous state.

Definition 3: The set $C_i \subseteq Q \times X^i$ is defined as follows:

- $\boldsymbol{i = 0}$: Initially, the length of the considered input sequences is $i = 0$. We define
$$C_0 := \big\{ \big(q, ()\big) \;\big|\; q \in Q,\; q \not\models \psi \big\}.$$
- $\boldsymbol{i \rightarrow i + 1}$: Based on C_i, C_{i+1} is computed:
$$C_{i+1} := \Big\{ \big(q, (x, \tilde{x})\big) \;\Big|\; q \in Q, x \in X, \tilde{x} \in X^i; \\ \forall q' \in Q\colon \Big(\big((q, x, q') \in R_E\big) \rightarrow \big((q', \tilde{x}) \in C_i\big) \Big) \Big\}$$

Obviously, C_0 satisfies the invariant defined above.

The invariant for C_{i+1} can be proven inductively based on the invariant for C_i: Let us consider some state $q \in Q$, some input $x \in X$, and some input sequence $\tilde{x} \in X^i$. By definition of C_{i+1}, $\big(q, (x, \tilde{x})\big)$ is in C_{i+1} iff for every possible successor $q' \in Q$ with $(q, x, q') \in R_E$: $(q', \tilde{x}) \in C_i$. This means that, regardless of which possible transition under x is taken, we arrive at some successor state q' such that each possible path starting with q' and resulting from the remaining input sequence $\tilde{x}$ ends in an erroneous state (by inductive assumption on C_i). Altogether, starting with q, each possible path resulting from the input sequence $(x, \tilde{x})$ (with length $i{+}1$) will end in an erroneous state.

The opposite direction of the invariant for C_{i+1} can be shown by induction in a similar manner.

Now, a uniform counterexample with length i exists, iff a sequence *starting with an initial state* is included in the set C_i.

The computation of uniform counterexamples is applied after the Black Box model checker produced the result that

[2] For ease of explanation, we assume that ψ contains only state variables. This is not a necessary restriction to our method, actually ψ may contain arbitrary signals of the design.

the safety property is violated for every possible Black Box substitution. That means that there is a (possibly non-uniform) counterexample of length d (where d is the number of backward steps that were necessary to prove unrealizability).

However, it may be possible that there is no uniform counterexample of length d (we will explain this issue in detail in Sect. III-F). Due to that, it is reasonable to look also for uniform counterexamples that are longer than d (at least by some small constant factor $c > 1$). We are not interested in uniform counterexamples much longer than d, since it is our goal to *simplify* counterexamples by our Black Box techniques. This upper bound on the length of uniform counterexamples provides our stop criterion for the iterative computation also in cases where there is no uniform counterexample at all.

Altogether, this leads to the algorithm shown in Fig. 4.

Theorem 1: The algorithm shown in Fig. 4 computes — if possible at all — a uniform counterexample (according to Def. 2) with minimum length $\leq c \cdot d$.

The proof of the theorem follows immediately from the arguments given above.

C. Example

We again use our example given in Figures 1 and 2 together with the safety property $\varphi = AG(\neg q_0 \vee \neg q_1)$. The possible transition relation R_E for this example is given in Section II.

Model checking for incomplete designs proves that φ is not satisfied for any substitution of the Black Box. For uniform counterexample computation, we evaluate C_i as defined above for increasing counterexample length i. For $i = 0$, C_0 contains all states directly violating the safety property $\psi = \neg q_0 \vee \neg q_1$ plus an empty input sequence:

$$C_0 = \big\{ \big(q_1 q_0, ()\big) \big\}$$

In the next step, we iteratively increase the length of the input sequence:

$$C_1 = \big\{ \big(q_1 \overline{q}_0, (x)\big), \big(\overline{q}_1 q_0, (\overline{x})\big), \big(\overline{q}_1 q_0, (x)\big) \big\}$$

Note that e.g. $\big(\overline{q}_1 \overline{q}_0, (x)\big) \notin C_1$ due to $\big(\overline{q}_1 \overline{q}_0, x, q_1 \overline{q}_0\big) \in R_E$ yet $\big(q_1 \overline{q}_0, ()\big) \notin C_0$. We continue:

$$\begin{aligned} C_2 = \big\{ & \big(q_1 \overline{q}_0, (\overline{x}, \overline{x})\big), \big(q_1 \overline{q}_0, (\overline{x}, x)\big), \\ & \big(q_1 q_0, (\overline{x}, x)\big), \big(q_1 q_0, (x, x)\big) \big\} \\ C_3 = \big\{ & \big(\mathbf{\overline{q}_1 \overline{q}_0}, (\mathbf{x}, \mathbf{\overline{x}}, \mathbf{x})\big), \big(q_1 \overline{q}_0, (x, \overline{x}, x)\big), \big(q_1 \overline{q}_0, (x, x, x)\big), \\ & \big(\overline{q}_1 q_0, (\overline{x}, \overline{x}, x)\big), \big(\overline{q}_1 q_0, (\overline{x}, x, x)\big), \big(\overline{q}_1 q_0, (x, \overline{x}, x)\big), \\ & \big(\overline{q}_1 q_0, (x, x, x)\big), \big(q_1 q_0, \overline{x}, \overline{x}, \overline{x}\big), \big(q_1 q_0, (\overline{x}, \overline{x}, x)\big), \\ & \big(q_1 q_0, (x, \overline{x}, \overline{x})\big), \big(q_1 q_0, (x, \overline{x}, x)\big) \big\} \end{aligned}$$

Since there is an input sequence for the initial state $\overline{q}_1 \overline{q}_0$ in C_3, we have found the uniform counterexample $(1, 0, 1)$ with minimum length $i = 3$.

D. Impact of different modeling of unknowns

Of course, uniform counterexamples depend on the set of possible transitions derived from the system. However, our approach (following [15]) considers different possibilities to model the behavior of Black Boxes in their environment leading to different approximations of the sets of possible transitions (see also Section II). The least expensive method computes possible transitions based on $(0, 1, X)$-logic; it leads to the largest number of possible transitions. Yet, due to well-known deficiencies of $(0, 1, X)$-logic, ternary logic may lead to "possible transitions" which are not present for any replacement of the Black Boxes (see also [15]), i.e., potentially possible paths are considered which are not present in any implementation. This "overapproximation" potentially prevents us from finding uniform counterexamples. In this case we have to consider more accurate approximations as discussed in Sect. II.

E. Symbolic Computation

For symbolic computation, we use the symbolic transition relation $\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}\,')$ as introduced in Sect. II and the characteristic function $\chi_{Sat(\neg\psi)}(\vec{q})$ of the set of states violating ψ. The uniform counterexamples can then be symbolically computed as shown in Fig. 5.

For our implementation we use symbolic representations based on AIGs [8].

F. Bounds on the Length of Uniform Counterexamples

The following Theorem 2 justifies our decision to abort the search for a uniform counterexample whenever its length exceeds its non-uniform counterpart by a constant factor. The proof of Theorem 2 presents an example where the size of the shortest uniform counterexample is much larger than the size of the shortest non-uniform counterexample. (Whereas this is an interesting theoretical result, this does not mean that such examples will be observed in practical applications.)

Theorem 2: There are incomplete circuits with n states where the length of the shortest uniform counterexample is in $\Omega(2^{\sqrt{n}})$.

Proof: Let $p = (2, 3, 5, 7, \dots)$ be the sequence of prime numbers.

Consider the automaton shown in Figure 6. Starting from the initial state q_0 and for input $x = 1$, there are m successors

$$\begin{aligned} & i := 0; \\ & C_0 := \big\{ \big(q, ()\big) \mid q \in Q,\ q \not\models \psi \big\}; \\ & \text{while } \big((\nexists q \in Q, \tilde{x} \in X^i\colon q \in I \wedge (q, \tilde{x}) \in C_i) \wedge (i < c \cdot d)\big)\ \{ \\ & \quad C_{i+1} := \Big\{ \big(q, (x, \tilde{x})\big) \mid q \in Q, x \in X, \tilde{x} \in X^i; \\ & \qquad\quad \forall q' \in Q\colon \Big(\big((q, x, q') \in R_E\big) \rightarrow \big((q', \tilde{x}) \in C_i\big) \Big) \Big\}; \\ & \quad i := i + 1; \\ & \} \end{aligned}$$

Fig. 4. Algorithm for uniform counterexample computation

$$\begin{aligned} & i := 0; \\ & \chi_{C_0}(\vec{q}) := \chi_{Sat(\neg\psi)}(\vec{q}); \\ & \text{while } \big((\chi_{C_i} \cdot \chi_I = \text{false}) \wedge (i < c \cdot d)\big)\ \{ \\ & \quad \chi_{C_{i+1}}(\vec{q}, \vec{x}_{i+1}, \dots, \vec{x}_1) := \forall \vec{q}\,' \Big(\big(\chi_{R_E}|_{\vec{x} \leftarrow \vec{x}_{i+1}}\big)(\vec{q}, \vec{x}_{i+1}, \vec{q}\,') \\ & \qquad\qquad \rightarrow \big(\chi_{C_i}|_{\vec{q} \leftarrow \vec{q}\,'}\big)(\vec{q}\,', \vec{x}_i, \dots, \vec{x}_1) \Big); \\ & \quad i := i + 1; \\ & \} \end{aligned}$$

Fig. 5. Symbolic algorithm for uniform counterexample computation

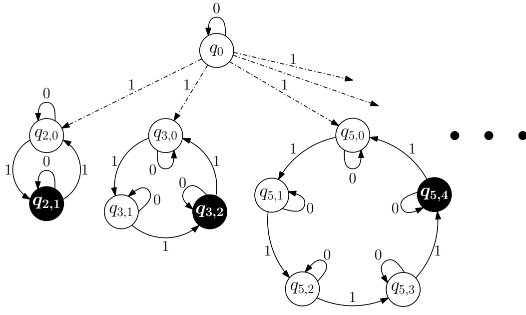

Fig. 6. Exemplary automaton for uniform counterexample length consideration. Fixed transitions are indicated by solid arrows, possible transitions are indicated by dashed and solid arrows. q_0 is the initial state and for every prime number p_i, q_{p_i,p_i-1} is an erroneous state.

$(q_{p_1,0},\ldots,q_{p_m,0})$ in different cycles with p_i states. It is easy to see that the number of states is $n = 1 + \sum_{i=1}^{m} p_i$.

In each of these cycles i of length p_i, the input $x = 1$ leads from $q_{p_i,j}$ to the next state in the cycle $q_{p_i,j+1(\text{mod } p_i)}$, while $x = 0$ always leads back to the same state. For each cycle, q_{p_i,p_i-1} is the only state violating the safety property. All transitions except the transitions for input $x = 1$ from q_0 to $q_{p_1,0},\ldots,q_{p_m,0}$ are fixed transitions.

Due to this construction, it can be seen that the shortest uniform counterexample has length $l = \prod_{i=1}^{m} p_i$ and always sets the input to 1.

The primorial $\prod_{i=1}^{m} p_i$ is known to be in $\Omega(2^{p_m})$. Together with

$$n = \sum_{i=1}^{m} p_i + 1 \leq \sum_{j=1}^{p_m} j = \frac{p_m^2 - p_m}{2} \leq p_m^2,$$

one can see that for this automaton with n states the shortest uniform counterexample has length $\Omega(2^{\sqrt{n}})$. ■

From a theoretical point of view, it is also interesting to consider upper bounds on the length of shortest uniform counterexamples:

Theorem 3: For any automaton corresponding to an incomplete circuit with n states the length of the shortest uniform counterexample is $\leq 2^n$.

Theorem 3 can be proven by a power set construction similar to the well-known transformation of non-deterministic finite automata to deterministic finite automata. Theorem 3 could be used as an alternative stop criterion for the algorithm of Sect. III-B, but remember that — from a practical point of view — we use Black Boxing as a means for simplifying counterexamples and thus, we are not interested in (uniform) counterexamples having huge lengths.

G. Lessons Learned for SAT Based Methods

The result of Theorem 2 is also interesting in the context of counterexample simplification in Bounded Model Checking [13]: It essentially says that there are examples where the number of unfoldings in BMC has to be exponentially enlarged before it is possible to 'simplify' the counterexample by 'lifting' all output signals of Black Boxes in all time frames (lifting means removing assignments to signals without losing the property that the remaining assignments imply a violation of the safety property).

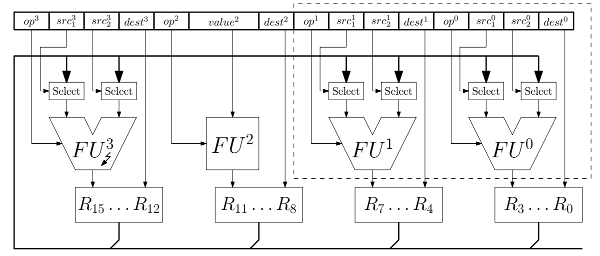

Fig. 7. A simple VLIW ALU with an error in the XOR operation in FU^0.

Moreover, note that our approach based on Black Box techniques is more powerful than SAT based methods for lifting assignments to signals, if the goal is to lift all output signals of Black Boxes in all time frames: SAT based methods start with a fixed counterexample and try to lift assignments. This specific counterexample may be only one out of a large number of possible counterexamples, and, by accident, it may be the case that lifting does not work for this specific counterexample, but it would work for other counterexamples. We will come back to this problem when considering our case study in Sect. IV.

By a formulation as a QBF (Quantified Boolean Formula) problem instead of a number of SAT problems it would be possible to check for all possible counterexamples whether the variables mentioned above may be lifted [25], [26], but QBF problems are harder to solve than SAT formulas and for QBF solving it could not yet be observed such a breakthrough as for SAT solving during recent years [27], [28].

IV. Case Study

As a case study, we will consider a simple VLIW ALU with four functional units as illustrated in Fig. 7; the VLIW instruction word (consisting of four parts for the four functional units) is considered to be the input of the design. Each functional unit $FU^0 \ldots FU^3$ is able to read from each of the 16 registers, while writing is limited to a local part of the registers (FU^0 may write to registers $R_0 \ldots R_3$, FU^1 to $R_4 \ldots R_7$ etc.).

FU^3 realizes the four logical instructions AND, OR, XOR and NOP, whereas the XOR operation is faulty and computes an OR instead (note that XOR and OR only differ when both inputs are 1). FU^2 implements a 'Load Immediate' operation (that writes a value given in the operation code into a specified register) and a NOP operation. FU^1 and FU^0 both realize the arithmetic operations ADD, SUB, MUL and NOP. All registers are assumed to be initialized by 0.

All our experiments were performed on a Dual Opteron running at 2 GHz and with 4 GB of RAM.

A. Complete Assignments and Lifting

This completely specified system can be checked with a property stating that if the last operation that FU^3 executed was "$R_{14} := R_{12} \oplus R_{13}$", then the current value of R_{14} has to be the current value of R_{12} XOR the current value of R_{13}.[3] Due to the erroneous XOR implementation, the property

[3]To turn this property into a *AG*-formula mentioning only current state bits, we had to cache the last operation in the design, so that it is available in the current state.

TABLE I

COUNTEREXAMPLE FOR THE CIRCUIT IN FIG. 7 BEFORE (ALL ENTRIES) AND AFTER LIFTING VARIABLE ASSIGNMENTS (BOLD ENTRIES ONLY).

Step	FU^3	FU^2	FU^1	FU^0
0	$R_{12} := R_0 \wedge R_0$	$\boldsymbol{R_8 := 0\ldots01}$	$R_4 := R_0 \cdot R_0$	$R_0 := R_0 \cdot R_0$
1	$\boldsymbol{R_{12} := R_8 \wedge R_8}$	$R_8 := 10\ldots0$	$\boldsymbol{R_4 := R_0 + R_8}$	$\boldsymbol{R_0 := R_8 - R_0}$
2	$\boldsymbol{R_{13} := R_0 \wedge R_4}$	$R_8 := 0\ldots00$	NOP	$R_0 := R_2 \cdot R_1$
3	$\boldsymbol{R_{14} := R_{12} \oplus R_{13}}$	$*$	$*$	$*$

TABLE III

UNIFORM COUNTEREXAMPLE FOR THE CIRCUIT IN FIG. 7 IN WHICH FU^1 AND FU^0 HAVE BEEN REPLACED BY A BLACK BOX.

Step	FU^3	FU^2	FU^1	FU^0
0	$*$	$R_9 := *\ldots*1$	$*$	$*$
1	$R_{12} := R_9 \vee R_9$	$*$	$*$	$*$
2	$R_{13} := R_9 \vee R_9$	$*$	$*$	$*$
3	$R_{14} := R_{12} \oplus R_{13}$	$*$	$*$	$*$

fails. It is now possible to compute a counterexample for the complete design, which includes a value for each primary input, thus a complete VLIW instruction in every step of the counterexample, by an arbitrary conventional counterexample computation method. Based on this assignment, it is possible to remove ('lift') some of these variable assignments using the lifting algorithm presented in [13].

As a representative example for such a counterexample computation method, we used a simple bounded model checker based on MiniSAT [28] that additionally implemented lifting. Table I shows a counterexample generated by BMC for the VLIW ALU with 12 Bit word width before lifting.

Based on this assignment, a reduced counterexample was obtained by lifting as many variables as possible; the bold entries in Tab. I illustrate this reduced counterexample. Obviously, still all functional units have to be considered in order to understand the counterexample.

Using this BMC tool, we performed experiments for several versions of the VLIW processor with differing word widths; Tab. II shows the overall run times for bounded model checking and lifting, the run times for lifting only, and the set of functional units the lifted counterexample argues about.

For comparison, Tab. II additionally gives the runtimes of bounded model checking using the VIS tool [29]; these runtimes do not include lifting, since a version of VIS including lifting was not at our disposal.

Note that BMC followed by lifting may or may not find uniform counterexamples as defined in this paper, since lifting starts with a *specific* counterexample found by a SAT solver.

In the following we will show that our method is able to provide a much better counterexample than that of Table I.

B. Counterexample Generation for Incomplete Designs

For our experiments, we replaced FU^1 and FU^0 by a Black Box (comp. the dashed box in Fig. 7).

Note that our first conjecture that we would need only FU^3 (which implements logical operations) to be kept in the design for deriving a minimized counterexample turned out to be incorrect: If FU^2, FU^1 and FU^0 are replaced by a Black Box, it is not possible to prove that the property fails.[4] Moreover note that the Black Box in Fig. 7 lies inside the cone of influence for the considered property, and thus can not be removed by a cone-of-influence reduction. Our method presented in Sect. III-E was able to find a uniform counterexample that included only instructions for FU^3 and FU^2. Table III shows the counterexample generated by our approach based on a symbolic computation of uniform counterexamples and a removal of nonessential input variable assignments. It is easy to see that this counterexample gives a much better and more succinct explanation for the error in the design than the counterexample from Tab. I, now arguing about a smaller number of components and input signals.

We performed experiments for several versions of the VLIW processor with varying bit widths. Whereas the symbolic model checker for incomplete designs that was used in [15] was based on the BDD package CUDD [30] and used relational preimage computation, we used an improved version in which boolean formulas are represented by And-Inverter-Graphs [8] and that used functional preimage computation [31], [32] for our experiments.

Our method was able to provide a counterexample for a word width of 64 bits in less than 11 minutes of CPU time. Detailed results for varying word widths can be found in Tab. IV. The runtimes given in the table cover the model checking for the incomplete design, the computation of a uniform counterexample and the computation of a complete

[4] It is easy to see that we need constants 1 in both operand registers R_{12} and R_{13} of the XOR operation in order to make the error observable. Using only FU^3, it is not possible to place these constants into R_{12} and R_{13}.

TABLE II

BOUNDED MODEL CHECKING COUNTEREXAMPLE GENERATION WITH LIFTING FOR THE CIRCUIT IN FIG. 7 WITH VARYING WORD WIDTHS.

word width	CPU sec Overall	CPU sec Lifting	Functional Units used in the counterexample	VIS CPU sec (No Lifting)
2	0.10	0.04	FU^3, FU^2, FU^0	0.1
4	0.20	0.06	FU^3, FU^2	0.2
6	0.34	0.10	FU^3, FU^2	0.4
8	0.52	0.16	FU^3, FU^2	0.6
12	1.44	0.74	FU^3, FU^2, FU^1, FU^0	1.2
16	2.12	1.02	FU^3, FU^2, FU^0	2.0
24	4.04	1.74	FU^3, FU^2	4.3
32	8.32	4.20	FU^3, FU^2, FU^0	8.7
48	19.80	9.66	FU^3, FU^2, FU^0	35.2
64	46.13	22.60	FU^3, FU^2, FU^0	83.3

TABLE IV

UNIFORM COUNTEREXAMPLE GENERATION FOR THE CIRCUIT IN FIG. 7 IN WHICH FU^1 AND FU^0 HAVE BEEN REPLACED BY A BLACK BOX FOR VARYING WORD WIDTHS.

word width	AIGs CPU sec	AIGs # Nodes	BDDs CPU sec	BDDs # Nodes
2	0.68	2077	20.58	203175
4	1.18	3703	846.98	2739457
6	2.02	5329	23.24	195284
8	2.97	6955	33.43	210342
12	5.18	10207	458.72	875048
16	10.81	13459	472.90	839244
24	27.16	19963	3273.05	2159946
32	34.08	26467	2488.55	1928284
48	165.45	39475	5363.69	4678588
64	638.95	52483	3601.53	2086592

simulation run demonstrating the error.

For comparison, Tab. IV shows also run times and node counts for a version of our model checker where AIGs were replaced by BDDs (CUDD [30]) for symbolic representations. The table shows a drastic increase of run times for the BDD based version compared to the AIG based version.

It is interesting to see that for our case study it is necessary to compute uniform counterexamples based on possible transition as defined in Def. 2 (Sect. III-A). The second approach discussed in Sect. III-A, which considered fixed transitions only, would not have been able to generate a counterexample, since no erroneous state is reachable by only fixed transitions in this example.

V. Conclusion and Future Work

We introduced a method to compute counterexamples for incomplete designs. This method can be used to compute more comprehensible counterexamples arguing about a reduced number of components in the system. Our method is based on the notion of 'uniform counterexamples' which are counterexamples showing an error effect without making assumptions on certain Black Boxes in the design. Experimental results, which demonstrate the usefulness of our concepts and algorithms, are complemented by theoretical results wrt. existence and potential lengths of uniform counterexamples.

For the future we plan to extend the approach from safety properties to more general classes of temporal formulas. Currently, the selection of parts of the design which will be replaced by Black Boxes is performed by the user based on the hierarchical structure of the system and the property at hand. We plan to develop methods which automate this selection process in order to obtain simplified counterexamples without or with a reduced user interaction.

References

[1] A. Sistla and E. Clarke, "The complexity of propositional linear temporal logics," *Journal of the ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[2] E. Clarke, E. Emerson, and A. Sistla, "Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[3] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, "Symbolic Model Checking: 10^{20} States and Beyond," *Information and Computation*, vol. 98(2), pp. 142–170, 1992.

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

[5] R. Bryant, "Graph - based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[6] M. Ganai, A. Gupta, and P. Ashar, "Efficient SAT-based unbounded symbolic model checking using circuit cofactoring," in *Int'l Conf. on Computer-Aided Design*, 2004, pp. 510–517.

[7] H.-J. Kang and I.-C. Park, "Sat-based unbounded symbolic model checking," *IEEE Trans. on CAD*, vol. 24, no. 2, pp. 129–140, February 2005.

[8] F. Pigorsch, C. Scholl, and S. Disch, "Advanced unbounded model checking based on aigs, bdd sweeping, and quantifier scheduling," in *Proceedings of the Conference on Formal Methods in Computer Aided Design (FMCAD)*. IEEE Computer Society Press, Nov 2006, pp. 89 – 96.

[9] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 1579. Springer Verlag, 1999.

[10] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Design Automation Conf.*, 1999.

[11] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties Using Induction and a SAT-solver," in *FMCAD*, ser. LNCS, W. H. Jr. and S. Johnson, Eds., vol. 1954. Springer, 2000, pp. 407–420.

[12] K. McMillan, "Interpolation and sat-based model checking," in *Computer Aided Verification*, vol. 2725. Springer Verlag, 2003, pp. 1–13.

[13] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in *TACAS*, vol. 2988. Springer, 2004, pp. 31–45.

[14] S. Shen, Y. Qin, and S. Li, "Minimizing counterexample with unit core extraction and incremental sat," in *VMCAI*, vol. 3385. Springer, 2005, pp. 298–312.

[15] T. Nopper and C. Scholl, "Approximate symbolic model checking for incomplete designs," in *Formal Methods in Computer-Aided Design*, ser. LNCS, A. J. Hu and A. K. Martin, Eds., vol. 3312. Austin, Texas: Springer Verlag, Nov 2004, pp. 290–305.

[16] C. Pixley, "A theory and implementation of sequential hardware equivalence," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 11, no. 12, pp. 1469–1478, 1992.

[17] T. Kropf and H.-J. Wunderlich, "A Common Approach to Test Generation and Hardware Verification Based on Temporal Logic," in *Int'l Test Conf.*, 1991, pp. 57–66.

[18] C.-J. H. Seger and R. E. Bryant, "Formal verification by symbolic evaluation of partially-ordered trajectories," *Formal Methods in System Design: An International Journal*, vol. 6, no. 2, pp. 147–189, March 1995.

[19] C.-J. H. Seger, R. B. Jones, J. W. O'Leary, T. F. Melham, M. Aagaard, C. Barrett, and D. Syme, "An industrially effective environment for formal hardware verification." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 9, pp. 1381–1405, 2005.

[20] R. Tzoref and O. Grumberg, "Automatic refinement and vacuity detection for symbolic trajectory evaluation." in *CAV*, ser. LNCS, T. Ball and R. B. Jones, Eds., vol. 4144. Springer Verlag, 2006, pp. 190–204.

[21] F. Copty, A. Irron, O. Weissberg, N. P. Kropp, and G. Kamhi, "Efficient debugging in a formal verification environment," in *Correct Hardware Design and Verification Methods (CHARME)*, ser. LNCS, T. Margaria and T. F. Melham, Eds., vol. 2144. Springer Verlag, September 2001, pp. 275–292.

[22] T. Nopper and C. Scholl, "Flexible modeling of unknowns in model checking for incomplete designs," in *8. GI/ITG/GMM Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, April 2005.

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

[24] O. Coudert, C. Berthet, and J. Madre, "Verification of synchronous sequential machines based on symbolic execution," in *Automatic Verification Methods for Finite State Systems*, ser. LNCS, vol. 407. Springer Verlag, 1989, pp. 365–373.

[25] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *Design Automation Conf.*, 2001, pp. 238–243.

[26] M. Herbstritt, B. Becker, and C. Scholl, "Advanced SAT-techniques for bounded model checking of blackbox designs," in *Proc. of Microprocessor Test and Verification Workshop (MTV)*. IEEE Computer Society, 2006.

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engeneering an efficient SAT solver," in *Design Automation Conf.*, 2001.

[28] N. Een and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing, 6th InternationalConference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8,2003 Selected Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2003, pp. 541–638.

[29] The VIS Group, "VIS: A system for verification and synthesis," in *Computer Aided Verification*, ser. LNCS, vol. 1102. Springer Verlag, 1996, pp. 428–432.

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

[31] T. Filkorn, "Functional extension of symbolic model checking," in *CAV '91: Proceedings of the 3rd International Workshop on Computer Aided Verification*. Springer, 1992, pp. 225–232.

[32] P. Williams, A. Biere, E. Clarke, and A. Gupta, "Combining decision diagrams and SAT procedures for efficient symbolic model checking," in *Computer Aided Verification*, ser. LNCS, vol. 1855. Springer Verlag, 2000, pp. 124–138.