Property Checking By Logic Relaxation

Eugene Goldberg
eu.goldberg@gmail.com

***Abstract*—We introduce a new framework for Property Checking (PC) of sequential circuits. It is based on a method called Logic Relaxation (LoR). Given a safety property, the LoR method relaxes the transition system at hand, which leads to expanding the set of reachable states. For j-th time frame, the LoR method computes a superset A_j of the set of bad states reachable in j transitions *only* by the relaxed system. Set A_j is constructed by a technique called partial quantifier elimination. If A_j does not contain a bad state and this state is reachable in j transitions in *the relaxed system*, it is also reachable in the original system. Hence the property in question does not hold.**

The appeal of PC by LoR is as follows. An inductive invariant (or a counterexample) generated by LoR is a result of computing the states reachable only in the relaxed system. So, the complexity of PC can be drastically reduced by finding a "faulty" relaxation that is close to the original system. This is analogous to equivalence checking whose complexity strongly depends on how similar the designs to be compared are.

I. Introduction

A. Motivation

Property checking is an important part of the formal verification of hardware. Recently, new powerful methods of property checking have been developed [10], [1]. A characteristic feature of those methods is that they use SAT-solving to avoid operating on quantified formulas e.g. performing quantifier elimination. This is done because of insufficient efficiency of the current algorithms for quantified logic. On the other hand, such algorithms have great potential because reasoning on quantified formulas facilitates very powerful transformations preserving equi-satisfiability rather than equivalence.

To address the problem of reasoning on quantified formulas, we have been developing a machinery of Dependency sequents (D-sequents) [5], [6], [9]. In particular, we have introduced a technique called Partial Quantifier Elimination (PQE) [8] that can boost the performance of algorithms operating on quantified formulas. Our research on D-sequents and PQE is still work in progress and we believe that catching up with SAT-based algorithms is just a matter of time. So we try to combine work on improving PQE algorithms with research that explains the benefits of such algorithms for formal verification [7], [3], [4]. In particular, in [4], we introduced a new verification method called Logic Relaxation (LoR) enabled by PQE. We showed that applying the LoR method to equivalence checking of combinational circuits facilitates generation of powerful inductive proofs. In this paper, we continue this work by applying the LoR method to property checking of sequential circuits.

B. Problem formulation

Let $M(S,X,Y,Z,S')$ be a sequential circuit where X, Y and Z are sets of input, internal and output combinational variables respectively, S and S' are sets of present and next state variables respectively. Let $T(S,X,Y,S')$ be a formula representing the transition relation specified by M. All formulas we consider in this paper are Boolean. We will assume that every formula is represented in the Conjunctive Normal Form (CNF). We will call a complete assignment $\boldsymbol{s}$ to state variables a $\boldsymbol{state}$. Henceforth, by an assignment $\boldsymbol{v}$ to a set of variables V we mean a *complete* assignment unless otherwise stated. Denote by ξ the transition system specified by transition relation T and a set of initial states $I(S)$. Let $P(S)$ specify the property of ξ to be verified. Given a Boolean formula $A(S)$, a state $\boldsymbol{s}$ is called an A-state if $A(\boldsymbol{s}) = 1$. We will refer to P-states and $\overline{P}$-states as **good** and **bad** ones respectively. In this paper, we consider checking a safety property. That is, given a property P, one needs to prove that a) no bad state of ξ is reachable from an I-state or b) a counterexample exists.

C. Property checking by logic relaxation

Denote by ξ^{rlx} a "relaxed" version of system ξ . Both ξ and ξ^{rlx} have the same set of initial states but are different in their transition relations. Let S_j, X_j, Y_j denote sets of variables of ξ in j-th time frame. Let $T_{j,j+1}$ denote formula $T(S_j,X_j,Y_j,S_{j+1})$ i.e. the transition relation of ξ in j-th time frame. Let $T^{rlx}_{j,j+1}$ denote formula $T^{rlx}(S_j,X_j,Y_j,S_{j+1})$ specifying the transition relation of ξ^{rlx} in j-th time frame. Formula $T_{j,j+1}$ implies $T^{rlx}_{j,j+1}$, so the set of transitions allowed in ξ^{rlx} is a superset of those in ξ.

The idea of Property Checking (PC) by Logic Relaxation (LoR) is as follows. Since the set of valid traces of ξ^{rlx} is a superset of that of ξ, a state reachable in j-th time frame of ξ is also reachable in ξ^{rlx}. Suppose that one has computed a set containing all states reachable in j-th time frame *only* in ξ^{rlx} . Then the existence of a bad state $\boldsymbol{s}$ that is not in this set and is reachable in the *relaxed* system ξ^{rlx} means that $\boldsymbol{s}$ is reachable in ξ as well and property P fails.

D. Boundary formulas

A key part of PC by LoR is computing so-called boundary formulas computing supersets of states reachable only in ξ^{rlx}. Formula $H_j(S_j)$ is called **boundary** for the pair (ξ , ξ^{rlx}) if it

- evaluates to 0 for every state that is reachable in ξ^{rlx} but not in ξ in j transitions
- evaluates to 1 for every state that is reachable in system ξ (and hence in ξ^{rlx}) in j transitions

The value of H_j is not specified for a state that is unreachable in ξ^{rlx} (and hence in ξ) in j transitions. On the one hand, H_j can be viewed as a "boundary" between sets of states reachable in ξ and ξ^{rlx} in j transitions, hence the name. On the other hand, since every $\overline{H_j}$-state is unreachable in ξ in j transitions, the H_j-states form an over-approximation of the set of states reachable in ξ in j transitions.

E. Transition relation relaxation

Let us show how one can use transition relation relaxation to build boundary formula H_1. The latter gives an over-approximation of the set of states reachable in ξ in one transition. Suppose that no bad state is reachable from an I-state of ξ in one transition. Let $\boldsymbol{s}$ be a bad state. Since $\boldsymbol{s}$ is unreachable from an I-state in one transition, formula $I_0 \wedge T_{0,1} \wedge C_{\boldsymbol{s}}$ is unsatisfiable. Here I_0 denotes $I(S_0)$ and $C_{\boldsymbol{s}}$ is the longest clause falsified by $\boldsymbol{s}$. (A **clause** is a disjunction of literals). Let us relax $T_{0,1}$ to make state $\boldsymbol{s}$ reachable. This means finding a formula $T^{rlx}_{0,1}$ implied by $T_{0,1}$ that makes $I_0 \wedge T^{rlx}_{0,1} \wedge C_{\boldsymbol{s}}$ satisfiable. Let $R_{0,1}$ be a formula such that $T_{0,1} \equiv T^{rlx}_{0,1} \wedge R_{0,1}$ i.e. $R_{0,1}$ specifies the "difference" between the transition relations. In the simplest case, $R_{0,1}$ is just a subset of clauses of $T_{0,1}$ and so $T^{rlx}_{0,1}$ is obtained from $T_{0,1}$ by removing the clauses of $R_{0,1}$. (In this paper, we use the notion of a CNF formula and that of a set of clauses interchangeably.)

Boundary formula H_1 is built by excluding states reachable only by the relaxed system ξ^{rlx}, specified by $T^{rlx}_{0,1}$, in one transition. Initially, H_1 is an empty set of clauses that represents a constant 1. Let $G(S_1)$ be a formula such that $\exists W_0[I_0 \wedge T^{rlx}_{0,1} \wedge R_{0,1}] \equiv G \wedge \exists W_0[I_0 \wedge T^{rlx}_{0,1}]$ where $W_0 = X_0 \wedge Y_0 \wedge S_0$. Finding G comes down to solving the Partial Quantifier Elimination (PQE) problem. (Only a part of the quantified formula leaves the scope of quantifiers, hence the name.) States falsifying G is a superset of states reachable with transition relation $T^{rlx}_{0,1}$ but not with $T_{0,1}$. In particular, G is falsified by state $\boldsymbol{s}$. The clauses of G are added to H_1. If $H_1 \rightarrow P$ holds, then H_1 is an over-approximation of the set of states reachable in ξ in one transition. Otherwise, there is a bad state $\boldsymbol{s}$ for which formula $I_0 \wedge T^{rlx}_{0,1} \wedge H_1 \wedge C_{\boldsymbol{s}}$ is unsatisfiable. Then the procedure above is applied again. That is transition relation $T^{rlx}_{0,1}$ is relaxed even more and a new formula G falsified by $\boldsymbol{s}$ is derived that makes up for this new relaxation. The set of clauses of G is added to H_1. This goes on until $H_1 \rightarrow P$ holds.

F. A high-level view of PC_LoR

In this paper, we formulate a an algorithm of PC by LoR called *PC_LoR*. To check if a property P holds, *PC_LoR* computes a sequence of boundary formulas $H_1, \dots, H_j$ that satisfy properties similar to those maintained in IC3 [1]. However these formulas are derived by employing transition relation relaxation and PQE rather than inductive clauses. Maintaining IC3-like properties is just a convenient way to guarantee that *PC_LoR* converges. If P holds in ξ, then eventually logically equivalent boundary formulas H_j and H_{j+1} are produced, meaning that H_j is an inductive invariant. Otherwise, *PC_LoR* fails to build a boundary formula H_j implying property P and finds a counterexample instead.

We also describe a version of *PC_LoR* that combines LoR with derivation of inductive clauses [1]. In IC3, a formula F_j over-approximating the set of states reachable in j transitions is built by tightening P. This tightening is done by adding to F_j inductive clauses excluding F_j-states from which a bad state is reachable in one transition. Such an approach may converge too slowly if an inductive invariant is "far" from property P. The idea of combining LoR with derivation of inductive clauses is as follows. The original boundary formula H_i is built by relaxation. The future corrections of H_i (done to maintain the IC3-like properties we mentioned above) are performed by tightening H_i up by inductive clauses. Such an approach can drastically speed up building an inductive invariant that is far from the property.

G. Merits of PC by LoR

This paper is motivated by some nice features of PC by LoR listed below. Since PC by LoR heavily relies on existence of efficient PQE solvers, realization of these features requires a boost in the performance of current PQE algorithms. We believe that this can be achieved via implementing some crucial techniques [4] that PQE solvers still lack. So getting the required performance of PQE is just a matter of time.

Our interest in PC by LoR is twofold. First, PC by LoR derives an inductive invariant (or a counterexample) by computing the difference between the original and relaxed transition systems. So, in a sense, the complexity of PC becomes *relative* since it depends on how different the original and relaxed systems are. This is analogous to equivalence checking whose complexity strongly depends on how similar the designs to be compared are. Second, by using a particular relaxation scheme one can take into account system and property structure/semantics. Suppose, for instance, that one needs to check a property P of a system ξ induced by interaction of two its subsystems ξ' and ξ''. Intuitively, an inductive invariant can be constructed by computing the difference between ξ and a relaxed system obtained from ξ by removing the interaction between ξ' and ξ''. If P holds, then bad states are reachable only in the relaxed system. That is the knowledge of problem semantics may help to generate an inductive invariant faster. We show how this idea works for equivalence checking (Section II).

H. Contributions and structure of the paper

The contribution of this paper is threefold. First, we introduce a new framework for PC. It is based on the idea of using transition relation relaxation and PQE to build an over-approximation of the set of reachable states. Second, we formulate a PC algorithm based on this idea and prove its correctness. Third, we formulate a PC algorithm combining transition relation relaxation with the machinery of inductive clauses.

The remainder of the paper is structured as follows. An example of PC by LoR is described in Section II. Basic definitions are given in Section III. Boundary formulas are discussed in Section IV. We describe *PC_LoR* in Section V. Section VI discusses two important modifications of *PC_LoR*. One of these modifications describes combining LoR with the machinery of inductive clauses. Some conclusions are given in Section VII.

II. An Example

In this section, we consider a special case of PC: equivalence checking of two identical sequential circuits. In Subsection II-A, we describe the example we consider. The problems with solving this example by interpolation and IC3 are discussed in Subsection II-B. Application of PC by LoR to this example is described in Subsection II-C. In particular, such application shows that by picking transition relation relaxation one can tailor a PC algorithm to the problem at hand.

A. Example description

Let $T^N(X^N, Y^N, S^N, S'^N)$ be a formula specifying the transition relation of a sequential circuit N. Here X^N, Y^N are the sets of input and internal variables of N respectively and S^N, S'^N are the sets of present and next state variables of N respectively. Let I^N be a formula specifying the initial states of N. Let circuit K be an identical copy of N. Let $T^K(X^K, Y^K, S^K, S'^K)$ and I^K be formulas specifying the transition relation and initial states of K. Suppose that one needs to verify equivalence of K and N defined as follows. K and N produce the same sequence of outputs for an identical sequence of values of X^N and X^K if they start in the same I-state.

The equivalence of N and K can be checked via building a sequential circuit M called a **miter** that is composed of N and K as shown in Figure 1. Let $T(X, Y, S, S') = T^N \wedge T^K \wedge EQ(X^N, X^K)$ where $X = X^N \cup X^K$, $Y = Y^N \cup Y^K$, $S = S^N \cup S^K$, $S' = S'^N \cup S'^K$. Given assignments $\boldsymbol{x^N}$ and $\boldsymbol{x^K}$ to X^K and X^N respectively, $EQ(\boldsymbol{x^N}, \boldsymbol{x^K}) = 1$ iff $\boldsymbol{x^N} = \boldsymbol{x^K}$. Formula T specifies the transition relation of miter M. Formula $I(S^N, S^K)$ specify the initial states of miter M where $I(\boldsymbol{s^N}, \boldsymbol{s^K}) = 1$ iff $\boldsymbol{s^N} = \boldsymbol{s^K}$ and $I^N(\boldsymbol{s^N}) = I^K(\boldsymbol{s^K}) = 1$. Note that the output variable z of M evaluates to 1 in j-th time frame iff N and K produce different assignments to output variables Z^N and Z^K. So proving the equivalence of N and K comes down to showing that the output z of miter M evaluates to 0 in every time frame. This can be done by proving that the following property $P(S^N, S^K)$ of M holds. $P(\boldsymbol{s^N}, \boldsymbol{s^K}) = 1$ iff N and K produce the same outputs in states $\boldsymbol{s^N}$ and $\boldsymbol{s^K}$ for every assignment $\boldsymbol{x^N}$ to X^N and $\boldsymbol{x^K}$ to X^K such that $\boldsymbol{x^N} = \boldsymbol{x^K}$.

Since circuits N and K are identical, the output z of M evaluates to 0 for every state $\boldsymbol{s} = (\boldsymbol{s^N}, \boldsymbol{s^K})$ where $\boldsymbol{s^N} = \boldsymbol{s^K}$. So $EQ(S^N, S^K) \rightarrow P$. However, in general, the reverse implication does not hold because N and K can produce the same output even in a state $\boldsymbol{s}$ where $\boldsymbol{s^N} \neq \boldsymbol{s^K}$. Note that

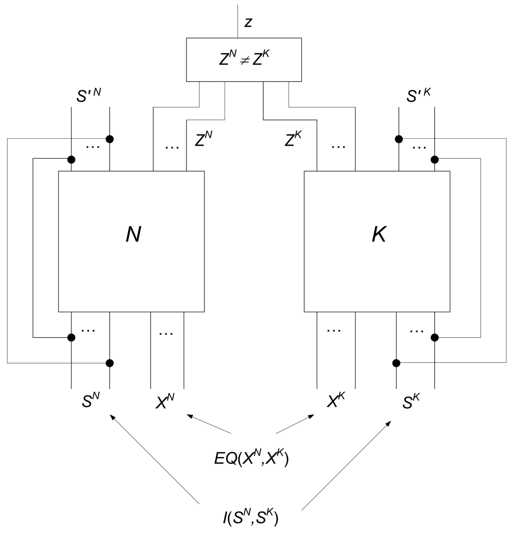

Fig. 1. Miter M of sequential circuits N and K

$EQ(S^N, S^K) \wedge T \rightarrow EQ(S'^N, S'^K)$ holds. So $EQ(S^N, S^K)$ is an inductive invariant.

B. Solving example by interpolation and IC3

Our example can be trivially solved by a method that tries to prove equivalence of corresponding state variables of N and K. However, such a method is unrobust since it can not be extended to the case where N and K are structurally close but not identical (e.g. if N does not have state variables that are functionally equivalent to variables of K.) So it is interesting to analyze solving our example by a general method that does not use pre-processing to identify equivalent state variables.

One can argue that checking the equivalence of two identical circuits can be hard for an interpolation based method. The performance of such a method strongly depends on the quality of an interpolant extracted from a proof that P holds for a limited number of transitions. Such a proof is produced by a general-purpose SAT-solver based on conflict clause learning. A known fact is that such solvers generate proofs of poor quality on equivalence checking formulas [2], [4]. This leads to producing interpolants of poor quality and hence slow convergence.

IC3 builds an inductive invariant by tightening property P via adding inductive clauses. Intuitively, the convergence rate of such a strategy strongly depends on how "far" an inductive invariant is from P. Consider, for instance, the inductive invariant $EQ(S^N, S^K)$. In general, EQ can be arbitrarily far from P especially if the transition system specified by N is deep and N can produce the same output in different states.

C. Solving example by LoR

Let us consider how our example is solved by LoR. As we mentioned in Subsection I-C, the basic operation of PC by LoR is to compute a superset of the set of states reachable in j transitions only by the relaxed system. Importantly, this superset is different from the precise set of reachable states only by the states (bad or good) that are *unreachable* by the relaxed system and hence by the original system. The objective here is to make sure that this superset contains all the bad states. Let us show how this is done for our example for the initial time frame. Let ξ denote the transition system specified by miter M and initial set of states I. Recall that transition relation $T_{0,1}$ of ξ is specified by $T^N_{0,1} \wedge T^K_{0,1} \wedge EQ^X_0$ where EQ^X_0 denote $EQ(X^N_0, X^K_0)$.

Let the relaxed transition $T^{rlx}_{0,1}$ for the initial time frame be obtained by dropping the clauses of EQ^X_0 i.e. $T^{rlx}_{0,1} = T^N_{0,1} \wedge T^K_{0,1}$. Let ξ^{rlx} denote the version of ξ where $T_{0,1}$ is replaced with $T^{rlx}_{0,1}$. In ξ^{rlx}, one can apply different input assignments to N and K. So ξ^{rlx} can transition to states $\boldsymbol{s}$=($\boldsymbol{s^N}$,$\boldsymbol{s^K}$) where $\boldsymbol{s^N} \neq \boldsymbol{s^K}$ and thus potentially reach bad states in one transition. Let us compute a boundary formula H_1. As we mentioned in Subsection I-D,

- the $\overline{H_1}$-states specify a superset of the set of states reachable in one transition only in ξ^{rlx} and
- the H_1-states is an over-approximation of the set of states reachable in ξ in one transition.

As we show in Section IV, H_1 can be found as a formula for which $\exists W_0[I_0 \wedge T_{0,1}] \equiv H_1 \wedge \exists W_0[I_0 \wedge T^{rlx}_{0,1}]$ holds where $W_0 = X_0 \wedge Y_0 \wedge S_0$. From Lemma 1 proved in the appendix it follows that formula H_1 equal to $EQ(S^N_1, S^K_1)$ satisfies the equality above. That is $EQ(S^N_1, S^K_1)$ can be used as a boundary formula H_1.

As we mentioned earlier, formula $EQ(S^K, S^N)$ is an inductive invariant. So by using a relaxed transition relation $T^{rlx}_{0,1}$ and building a boundary formula H_1 separating ξ and ξ^{rlx} one generates an inductive invariant. Such fast convergence is not a result of pure luck. The choice of relaxation above has a very simple explanation. Miter M consists of circuits N and K "interacting" with each other via combinational input variables. Circuits N and K interact correctly if the output of M is always 0. Intuitively, to verify that N and K interact correctly one needs to compute the difference between the original miter and a relaxed one where communication between N and K is cut off. If N and K are equivalent, miter M can produce output 1 only if N and K do not talk with each other. In Subsection VI-B, we argue that such relaxation can be successfully used in a general algorithm of sequential equivalence checking.

III. Basic Definitions

Definition 1: Let ξ be a transition system specified by transition relation $T(S,X,Y,S')$ introduced in Subsection I-B. A sequence of states ($\boldsymbol{s_m}$,...,$\boldsymbol{s_j}$) is called a **trace**. This trace is called **valid** if $\exists X \exists Y[T(\boldsymbol{s_k},X,Y,\boldsymbol{s_{k+1}})] = 1$, $k = m,\dots,j-1$.

Definition 2: Let I specify the initial states of system ξ. Given a property P of ξ, a valid trace ($\boldsymbol{s_0}$,...,$\boldsymbol{s_j}$) is called a **counterexample** if $I(\boldsymbol{s_0}) = 1$, $P(\boldsymbol{s_k}) = 1$,$k = 0,\dots,j-1$,$P(\boldsymbol{s_j}) = 0$.

Definition 3: Let ξ and η be two transition systems depending on the same set of variables S,X,Y,S'. We will say that η is a **relaxation** of system ξ if the set of valid traces of the former is a superset of that of the latter.

Definition 4: Let ξ be a system specified by transition relation T and formula I specifying initial states. Denote by $\boldsymbol{\xi^{rlx}_j}$ a relaxation of ξ such that

- ξ and ξ^{rlx}_j have identical sets of initial states and
- $T_{k,k+1} \equiv T^{rlx}_{k,k+1}$, $k \neq j$ and $T_{j,j+1} \rightarrow T^{rlx}_{j,j+1}$

In this paper, by a quantified formula we mean one with *existential* quantifiers. Given a quantified formula $\exists W[A(V,W)]$, the problem of quantifier elimination is to find a quantifier-free formula $A^*(V)$ such that $A^* \equiv \exists W[A]$. Given a quantified formula $\exists W[A(V,W) \wedge B(V,W)]$, the problem of **Partial Quantifier Elimination** (**PQE**) is to find a quantifier-free formula $A^*(V)$ such that $A^* \wedge \exists W[B] \equiv \exists W[A \wedge B]$. Note that formula B remains quantified (hence the name *partial* quantifier elimination). We will say that formula A^* is obtained by **taking $\boldsymbol{A}$ out of the scope of quantifiers** in $\exists W[A \wedge B]$. Importantly, there is a strong relation between PQE and the notion of *redundancy* of a clause in a quantified formula. For instance, solving the PQE problem above comes down to finding a set of clauses $A^*(V)$ implied by $A \wedge B$ that makes the clauses of A redundant in $A^* \wedge \exists W[A \wedge B]$. That is $A^* \wedge \exists W[A \wedge B] \equiv A^* \wedge \exists W[B]$.

IV. Boundary Formulas

In this section, we present boundary formulas. In Subsection IV-A we define boundary formulas and explain their relation to PQE. Building boundary formulas inductively is described in Subsection IV-B.

A. Definition of boundary formulas and their relation to PQE

Definition 5: Let ξ^{rlx} be a relaxation of system ξ and P be a property of ξ. Formula H_j is called **boundary** for the pair (ξ, ξ^{rlx}) if

1) $H_j(\boldsymbol{s}) = 0$, for every state $\boldsymbol{s}$ that is reachable in ξ^{rlx} and unreachable in ξ in j transitions
2) $H_j(\boldsymbol{s}) = 1$, for every state $\boldsymbol{s}$ that is reachable in ξ (and hence in ξ^{rlx}) in j transitions

Boundary formula H_j specifies the set of states reachable only by ξ^{rlx} i.e. separates ξ and ξ^{rlx} (hence the name "boundary"). We will say that H_j is just a boundary formula if the corresponding relaxation is obvious from the context.

Proposition 1 below gives a sufficient condition for a formula to be boundary. Let system ξ^{rlx}_j be obtained by relaxing only the transition relation of j-th time frame (see Definition 4). Let I_0 denote $I(S_0)$. Let $\mathbb{W}_{j-1}$ denote $W_0 \cup \dots \cup W_{j-1}$ where $W_i = S_i \cup X_i \cup Y_i$, $i = 0,\dots,j-1$. Let $\mathbb{T}_j$ denote $T_{0,1} \wedge \dots \wedge T_{j-1,j}$. Let $\mathbb{T}^{rlx}_j$ denote $\mathbb{T}_{j-1} \wedge T^{rlx}_{j-1,j}$.

Proposition 1: Let H_j be a formula (depending only on variables of j-th cut) such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$. Then H_j is a boundary formula for the pair (ξ, ξ^{rlx}_j).

Proofs of the propositions are given in the appendix.

Proposition 2: Let $T_{j-1,j} = T^{rlx}_{j-1,j} \wedge R_{j-1,j}$. Let H_j be a formula such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j \wedge R_{j-1,j}] \equiv H_j \wedge$

$\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j^{rlx}]$. Then H_j is a boundary formula for the pair (ξ, ξ_j^{rlx}).

One can view $R_{j-1,j}$ as a formula specifying the "difference" between $T_{j-1,j}$ and $T_{j-1,j}^{rlx}$ Proposition 2 suggests that H_j can be obtained by taking $R_{j-1,j}$ out of the scope of quantifiers i.e. by PQE.

B. Building boundary formulas inductively

Proposition 1 suggests that adding a boundary formula H_j makes up for the difference between $T_{j-1,j}$ and $T_{j-1,j}^{rlx}$. Suppose that one relaxes transition relation in every time frame. Let $\mathbb{T}_j^{RLX}$ denote $T_{0,1}^{rlx} \wedge \cdots \wedge T_{j-1,j}^{rlx}$. Let $\mathbb{H}_j$ denote $H_0 \wedge \cdots \wedge H_j$ where $H_0 = I$ and $H_1, \dots, H_j$ are boundary formulas. Then the following proposition is true.

Proposition 3: $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \equiv \exists \mathbb{W}_{j-1}[\mathbb{H}_j \wedge \mathbb{T}_j^{RLX}]$.

Boundary formulas $H_0, \dots, H_m$ can be built by induction using the following procedure. Let $T_{j-1,j} = T_{j-1,j}^{rlx} \wedge R_{j-1,j}$, $j = 1, \dots, m$. (We assume that $R_{j-1,j}$ is different in different time frames.) Formula $H_0 = I$ and formula H_j, $0 < j \leq m$ is obtained by taking $R_{j-1,j}$ out of the scope of quantifiers in formula $\exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}_j^{RLX} \wedge R_{j-1,j}]$. That is $\exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}_j^{RLX} \wedge R_{j-1,j}] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}_j^{RLX}]$. The correctness of this procedure follows from Proposition 4 of the appendix.

Note that the greater k, the larger the formula in which $R_{k-1,k}$ is taken out of the scope of quantifiers. This topic is discussed in [4]. There we argue the following. In [8], we introduced a PQE algorithm based on the machinery of D-sequents [5], [6]. The growth of formula size mentioned above will cripple the performance of the algorithm of [8] since the latter lacks a few crucial techniques e.g. D-sequent re-using. However, if a PQE solver employs D-sequent re-using, this problem will either go away completely or at least will be greatly mitigated.

V. An Algorithm Of PC By LoR

In this section, we describe an algorithm of PC by LoR called *PC_LoR*. This algorithm is meant only for systems that have the stuttering feature. In Subsection V-A, we explain the advantages of systems with stuttering and show how stuttering can be introduced by a minor modification of the system at hand if the latter does not have it. Subsections V-B, V-C, V-D describe the properties of boundary formulas maintained by *PC_LoR* to guarantee its convergence. A description of the pseudo-code of *PC_LoR* is given in Subsections V-E and V-F. The correctness of *PC_LoR* is proved in Subsection V-G.

A. Stuttering

Suppose that one needs to check that a property P of a sequential circuit M holds. Let T be the transition relation specified by M and ξ be the transition system defined by T and a formula I specifying the initial states (see Subsection I-B). The *PC_LoR* algorithm described in this section is based on the assumption that ξ has the **stuttering** feature i.e. ξ can stay in a given state arbitrarily long. This means that for every present state $\boldsymbol{s}$, there is an input assignment $\boldsymbol{x}$ such that the next state produced by circuit M is also $\boldsymbol{s}$. If ξ does not have this feature, one can introduce stuttering by adding to circuit M a combinational input variable v. The modified circuit M works as before if $v = 1$ and copies its current state to the output state variables if $v = 0$. On the one hand, introduction of stuttering does not affect the reachability of a bad state. On the other hand, stuttering guarantees that ξ has two nice properties. First, $\exists W[T(S, X, Y, S')] \equiv 1$ holds where $W = S \cup X \cup Y$. Indeed for every next state $\boldsymbol{s'}$, T specifies a "stuttering transition" from $\boldsymbol{s}$ to $\boldsymbol{s'}$ where $\boldsymbol{s} = \boldsymbol{s'}$. Second, if a state is unreachable in ξ after n transitions it is also unreachable after m transitions if $m < n$.

B. Four properties to guarantee convergence

The essence of *PC_LoR* is to build a boundary formula H_j for every time frame. Boundary formulas are generated by *PC_LoR* one by one. We assume that H_0 is set to I. Let $\mathbb{T}_j$ denote $T_{0,1} \wedge \cdots \wedge T_{j-1,j}$, $j > 0$ and $\mathbb{T}_0 \equiv 1$. We will refer to the four conditions below as **CO conditions** (where CO stands for Convergence of Over-approximations).

1) $I \rightarrow H_j$
2) $H_j \rightarrow P$,
3) $H_{j-1} \wedge T_{j-1,j}^{rlx} \rightarrow H_j$,
4) $H_{j-1} \rightarrow H_j$,

(When we write formulas like $I \rightarrow H_j$ we assume that the sets of variables are unified for the left and right parts of the implication. That is $I \rightarrow H_j$ actually means $I(S) \rightarrow H_j(S)$.) The CO conditions are similar to those imposed on formulas F_i specifying supersets of reachable states in IC3 [1]. However, formulas H_j are built via relaxation of transition relation and PQE i.e. quite differently from F_i of IC3. The convenience of the CO conditions is that no matter how formulas satisfying these conditions are built, eventually a counterexample or an inductive invariant are generated.

C. Providing first and second CO conditions

The first CO condition of Subsection V-B is achieved as follows. Formula H_j is built by resolving clauses of $I_0 \wedge \mathbb{T}_j$. So H_j is implied by $I_0 \wedge \mathbb{T}_j$. Due to the stuttering feature, this means that H_j is also implied by I alone.

The second CO condition is provided in two steps. Suppose that all boundary formulas up to H_{j-1} already satisfy the CO conditions and *PC_LoR* starts building formula H_j. In the first step, *PC_LoR* checks if $H_{j-1} \wedge T_{j-1,j} \rightarrow P$. If not, then there is an H_{j-1}-state $\boldsymbol{s_{j-1}}$ that reaches a bad state in one transition. *PC_LoR* tries to strengthen H_{j-1} by conjoining the latter with a CNF formula G falsified by $\boldsymbol{s_{j-1}}$. To derive this formula, *PC_LoR* calls procedure *RemBadSt* described in Subsection V-F. It either generates a trace leading to $\boldsymbol{s_{j-1}}$ (which means that P fails) or returns formula G above. Formula G is built by relaxing transition relations of some previous time frames even more and strengthening boundary formulas of those time frames to make up for such additional relaxation.

The second step starts when $H_{j-1} \wedge T_{j-1,j} \rightarrow P$ holds. In this step, *PC_LoR* calls procedure *FinRlx* that relaxes

```
PC_LoR(T, I, P) {
1  $H_0 := I$;
2  $j = 1$;
3  while (true) {
4    $T^{rlx}_{j-1,j} := T_{j-1,j}$;
5    $H_j := 1$;
6    $Cex := RemBadSt(\mathbb{H}_j, \mathbb{T}^{RLX}_j, P, j)$;
7    if ($Cex \neq nil$) return($No$);
8    $FinRlx(\mathbb{H}_j, \mathbb{T}^{RLX}_j)$
9    $ThirdCOcond(\mathbb{H}_j, \mathbb{T}^{rlx}_j)$;
10   $Inv := FinTouch(\mathbb{H}_j, \mathbb{T}^{rlx}_j)$ ;
11   if ($Inv$) return($Yes$);
12   $j := j + 1$; }}
```

Fig. 2. *PC_LoR* procedure

transition relation $T_{j-1,j}$ of $(j-1)$-th time frame and builds formula H_j implying P that makes up for relaxing $T_{j-1,j}$. Originally, $H_j = 1$ and $T^{rlx}_{j-1,j} = T_{j-1,j}$. If there is an H_j-state $\boldsymbol{s_j}$ that falsifies P, *PC_LoR* relaxes the current transition relation $T^{rlx}_{j-1,j}$ to make $\boldsymbol{s_j}$ reachable from an H_{j-1}-state. This relaxation has the form $T^{rlx}_{j-1,j} = T^{*rlx}_{j-1,j} \wedge R_{j-1,j}$ where $T^{rlx}_{j-1,j}$ is the current relaxed formula and $T^{*rlx}_{j-1,j}$ is a new one that makes $\boldsymbol{s_j}$ reachable. *PC_LoR* looks for a formula G such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{H}_j \wedge \mathbb{T}^{RLX}_{j-1,j} \wedge R_{j-1,j}] \equiv G \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{H}_j \wedge \mathbb{T}^{RLX}_{j-1,j}]$. Here $\mathbb{T}^{RLX}_{j-1,j}$ is equal to $\mathbb{T}^{RLX}_{j-2,j-1} \wedge T^{*rlx}_{j-1,j}$. Formula G is falsified by $\boldsymbol{s_j}$ and is conjoined with H_j to exclude this state. This goes on until H_j implies P.

D. Providing third and fourth CO conditions

After H_j is generated as described above, it satisfies the first two CO conditions of Subsection V-B but the third condition, in general, does not hold, i.e. $H_{j-1} \wedge T^{rlx}_{j-1,j} \not\rightarrow H_j$. This happens if a clause of an m-th time frame where $m < j-1$ is employed by procedure *FinRlx* above when generating H_j that implies P. Let $\boldsymbol{s_{j-1}}$ be an H_{j-1}-state that is one transition away from a state falsifying H_j. Then *PC_LoR* derives a formula falsified by $\boldsymbol{s_{j-1}}$ and conjoins it with H_{j-1}. This formula is derived by the procedure above used to eliminate H_{j-1}-states that are one transition away from a bad state. This goes on until $H_{j-1} \wedge T^{rlx}_{j-1,j} \rightarrow H_j$ holds. Even if the third condition $H_{m-1} \wedge T^{rlx}_{m-1,m} \rightarrow H_m$ holds for $m < j$, it may get broken after adding clauses to formula H_m. Then the procedure above is used to eliminate H_{m-1}-states that are one transition away from states falsifying H_m.

The fourth CO condition is very easy to maintain. Due to the stuttering feature, $\exists \mathbb{W}_{m-1}[I_0 \wedge \mathbb{T}_m] \rightarrow \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j]$, $m < j$. So every clause of H_j can be added to every boundary formula H_m, $m < j$.

E. Pseudo-code of PC_LoR

The pseudo-code of *PC_LoR* is given Figure 2. Boundary formulas are derived in the while loop (lines 3-12). In every iteration, a boundary formula H_j is derived and j is incremented by one. Originally, H_0 is set to I and j is set to 1.

Every iteration starts by making H_j satisfy the second CO condition i.e. $H_j \rightarrow P$ (lines 4-8). First *PC_LoR* calls

```
RemBadSt(H_j, T_j^RLX, P, j) {
1  $Cex := \emptyset$;
2  $length := 0$;
3  while (true) {
4    if ($length = 0$) {
5      $(\boldsymbol{s_{j-1}}, \boldsymbol{s_j}) := FndBadSt(H_{j-1} \wedge T^{rlx}_{j-1,j} \wedge \overline{P})$;
6      if ($(\boldsymbol{s_{j-1}}, \boldsymbol{s_j}) = nil$) return($nil$);
7      $Cex := (\boldsymbol{s_{j-1}}, \boldsymbol{s_j})$;
8      $length := 2$;
9      continue; }
   ----------
10   $k := j - length + 1$;
11   if ($k = 0$) return($Cex$);
12   $\boldsymbol{s_k} := FirstState(Cex)$;
13   $\boldsymbol{s_{k-1}} := ExtCex(H_{k-1} \wedge T^{rlx}_{k-1,k}, \boldsymbol{s_k})$;
   ----------
14   if ($\boldsymbol{s_{k-1}} \neq nil$) {
15     $Cex := (\boldsymbol{s_{k-1}}, Cex)$;
16     $length := length + 1$;
17     continue; }
   ----------
18   $R_{k-1,k} := Relax(H_{k-1}, T^{rlx}_{k-1,k}, s_k)$;
19   $PQE(R_{k-1,k}, H_k, T^{rlx}_{k-1,k}, \mathbb{H}_{k-1}, \mathbb{T}^{RLX}_{k-1})$;
20   $RemFrstSt(Cex, \boldsymbol{s_k})$;
21   $length := length - 1$;}}
```

Fig. 3. *RemBadSt* procedure

```
FinRlx(H_j, T_j^RLX) {
1  while (true) {
2    $\boldsymbol{s} := FindSat(H_j \wedge \overline{P})$;
3    if ($\boldsymbol{s} = nil$) return;
4    $R_{k-1,k} := Relax(H_{j-1}, T^{rlx}_{j-1,j}, s)$;
5    $PQE(R_{j-1,j}, H_j, T^{rlx}_{j-1,j}, \mathbb{H}_{j-1}, \mathbb{T}^{RLX}_{j-1})$;}}
```

Fig. 4. *FinRlx* procedure

procedure $RemBadSt$ that either returns a counterexample or strengthens formula H_{j-1} to guarantee $H_{j-1} \wedge T_{j-1,j} \rightarrow P$. Procedure $RemBadSt$ is described in detail in Subsection V-F. If $RemBadSt$ returns a counterexample Cex, *PC_LoR* terminates reporting that property P failed. Otherwise, *PC_LoR* calls procedure $FinRlx$ shown in Figure 4. Its work was described in Subsection V-C: $FinRlx$ relaxes the transition relation of $(j-1)$-th time frame and adds clauses making up for this relaxation to H_j until $H_j \rightarrow P$ holds.

To make sure that $H_{j-1} \wedge T^{rlx}_j \rightarrow H_j$ holds, *PC_LoR* calls procedure $ThirdCOcond$ that works as described in Subsection V-D. Finally, to guarantee that $H_{m-1} \rightarrow H_m$ holds for all $1 \leq m \leq j$, procedure $FinTouch$ is called. First, this procedure tries to push every clause C of H_j to previous boundary formulas. If C is not implied by H_m, $m < j$, it is added to H_m and $FinTouch$ tries to push C to H_{m-1}. Otherwise, the process of pushing clause C stops: if C is implied by H_m it is also implied by every formula H_k, $k < m$. The process of pushing clauses of H_j may break third CO condition for some boundary formulas. In this case, the $ThirdCOcond$ procedure is called to repair this condition. Eventually, $FinTouch$ makes all boundary formulas H_i, $i = 0, \ldots, j$ meet third and fourth CO conditions.

Procedure *FinTouch* also checks if $H_m \rightarrow H_{m-1}$ holds for some m, $0 \leq m \leq j$. If so, then $H_{m-1} \equiv H_m$ and H_{m-1} is an inductive invariant (see the proof of Proposition 6). Checking for presence of an inductive invariant by testing logical implication is harder than by checking syntactic equivalence performed in IC3. However, one can use optimization to mitigate this problem. Here is an example of such optimization. Formula H_m implies H_{m-1} iff every clause of H_{m-1} is implied by H_m. If a clause of H_{m-1} is implied by H_m, it remains implied no matter what clauses are added to H_{m-1} and H_m. So when checking if $H_m \rightarrow H_{m-1}$ holds, it suffices to check for implication every clause $C \in H_{m-1}$ that is not marked as implied by H_m yet. If C is implied by H_m, it is marked to avoid testing it in the future. Otherwise, $H_m \rightarrow H_{m-1}$ does not hold and no testing of other unmarked clauses of H_{m-1} is necessary.

F. Description of RemBadSt procedure

The pseudo-code of the *RemBadSt* procedure is given in Figure 3. The goal of *RemBadSt* is to strengthen boundary formula H_{j-1} so that $H_{j-1} \wedge T_{j,j-1} \rightarrow P$ holds. This is the first step of generation of formula H_j that implies P (see Subsection V-C). If H_{j-1} cannot be strengthened to guarantee the condition above, then a counterexample of length j is generated by *RemBadSt*.

All the work is done in a while loop (lines 3-21) where *RemBadSt* tries to construct a counterexample. This counterexample is built in reverse from a bad state reachable from an H_{j-1}-state. In every iteration of the loop, *RemBadSt* either extends the current trace by one more state or shows that the last H_m-state of the trace cannot be reached from an H_{m-1}-state. The latter triggers tightening up formula H_m after additional relaxation of the current transition $T^{rlx}_{m-1,m}$. The length of the current trace is specified by variable *length*. The body of the while loop can be partitioned into parts separated by the dotted lines in Figure 3. If *length* $= 0$, the current trace is empty and *RemBadSt* tries to initialize it (lines 5-9). Namely, it looks for an H_{j-1}-state $\boldsymbol{s_{j-1}}$ that is one transition away from a bad state $\boldsymbol{s_j}$. If such states $\boldsymbol{s_{j-1}}$ and $\boldsymbol{s_j}$ are found, counterexample *Cex* is initialized with ($\boldsymbol{s_{j-1}}$,$\boldsymbol{s_j}$). Otherwise, *RemBadSt* returns *nil* reporting that $H_{j-1} \wedge T_{j,j-1} \rightarrow P$ holds.

If *length* > 0, *RemBadSt* tries to extend the current trace (lines 10-13). Let $\boldsymbol{s_k}$ be the state added to *Cex* the last. If $k = 0$ i.e. if $\boldsymbol{s_k}$ is an I-state, the current *Cex* is a counterexample and *RemBadSt* terminates returning *Cex*. Otherwise, *RemBadSt* tries to find an H_{k-1}-state $\boldsymbol{s_{k-1}}$ that is one transition away from $\boldsymbol{s_k}$. If *RemBadSt* succeeds, *Cex* is extended by $\boldsymbol{s_{k-1}}$ (lines 15-16).

If *RemBadSt* fails to find $\boldsymbol{s_{k-1}}$, *Cex* cannot be extended to a counterexample. Then *RemBadSt* does the following (lines 18-21). The current transition relation $T^{rlx}_{k-1,k}$ is relaxed even more as described in Subsection V-C. Namely, $T^{rlx}_{k-1,k}$ is represented as $T^{*rlx}_{k-1,k} \wedge R_{k-1,k}$ where $T^{*rlx}_{k-1,k}$ is a new transition relation for $(k-1)$-th time frame that makes $\boldsymbol{s_k}$ reachable. *RemBadSt* calls a PQE-solver to build a

```
PC_LoR(T, I, P) {
  .....
4    $T^{rlx}_{j-1,j} := T_{j-1,j}$;
5*   $H_j := EducatGuessRlx(\mathbb{H}_{j-1}, \mathbb{T}^{RLX}_j)$;
6    $Cex := RemBadSt(\mathbb{H}_j, \mathbb{T}^{RLX}_j, P, j)$;
  .....
```

Fig. 5. *PC_LoR* plus relaxation by an educated guess

formula G such that $\exists \mathbb{W}_{k-1}[I_0 \wedge \mathbb{H}_k \wedge \mathbb{T}^{RLX}_{k-1,k} \wedge R_{k-1,k}] \equiv G \wedge \exists \mathbb{W}_{k-1}[I_0 \wedge \mathbb{H}_k \wedge \mathbb{T}^{RLX}_{k-1,k}]$. Here $\mathbb{T}^{RLX}_{k-1,k}$ is equal to $\mathbb{T}^{RLX}_{k-2,k-1} \wedge T^{*rlx}_{k-1,k}$. Formula G is falsified by $\boldsymbol{s_k}$ and so is conjoined with H_k to exclude this state. Then *RemBadSt* removes $\boldsymbol{s_k}$ from *Cex* and starts a new iteration.

G. Correctness of PC_LoR

This subsection lists propositions proving correctness of *PC_LoR*.

Proposition 5: Let H_j, $j = 1, \ldots, m$ be formulas derived by *PC_LoR* for m time frames where $H_j \rightarrow P$. Then property P holds for system ξ for at least m transitions.

Proposition 6: *PC_LoR* is sound.

Proposition 7: *PC_LoR* is complete.

VI. Two Important Modifications Of *PC_LoR*

In this section, we consider two modifications of the *PC_LoR* algorithm described in Section V. The first modification is to incorporate a "manual" relaxation that exploits the semantics of the system. The second modification is to combine LoR with the machinery of inductive clauses of IC3. Sequential equivalence checking is a promising application of the second modification.

A. Relaxation by an educated guess

In this subsection, we describe a modification of *PC_LoR* that starts building a boundary formula H_j by a relaxation that is just a guess tailored to a particular class of systems/properties. An example of such a relaxation is given in Section II. The pseudo-code of modified *PC_LoR* is given in Figure 5. The only difference between the original version shown in Fig. 2 and the modified one is in line 5 where function *EducatGuessRlx* is called instead of setting H_j to 1. This function does the following. First it represents the original transition relation $T_{j-1,j}$ as $T^{rlx}_{j-1,j} \wedge R_{j-1,j}$. Here $T^{rlx}_{j-1,j}$ is the relaxed transition relation replacing $T_{j-1,j}$. Then *EducatGuessRlx* calls a PQE solver to build a formula H_j such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_{j-1,j} \wedge R_{j-1,j}] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_{j-1,j}]$. Here $\mathbb{T}^{RLX}_{j-1,j}$ is equal to $\mathbb{T}^{RLX}_{j-2,j-1} \wedge T^{rlx}_{j-1}$.

B. Combining LoR with machinery of inductive clauses

In this subsection, we describe an algorithm called *LoR_IC* (IC stands for Inductive Clauses) that combines LoR and the machinery of inductive clauses introduced by IC3 [1]. Given a transition relation T, clause C is called **inductive** with respect to formula F if $F(S) \wedge C(S) \wedge T(S, X, Y, S') \rightarrow C(S')$ holds. Our interest in *LoR_IC* is twofold. First, when

```
LoR_IC(T, I, P) {
1   $H_0 := I$;
2   $j = 1$;
3   while ($true$) {
4     $T^{rlx}_{j-1,j} := T_{j-1,j}$;
5     $H_j := 1$;
6*    $Cex := RemBadSt_{IC}(\mathbb{H}_j, \mathbb{T}^{RLX}_j, P, j)$;
7     if ($Cex \neq nil$) return($No$);
8     $FinRlx(\mathbb{H}_j, \mathbb{T}^{RLX}_j)$
9*    $ThirdCOcond_{IC}(\mathbb{H}_j, \mathbb{T}^{rlx}_j)$;
10*   $Inv := FinTouch_{IC}(\mathbb{H}_j, \mathbb{T}^{rlx}_j)$ ;
11    if ($Inv$) return($Yes$);
12    $j := j + 1$; }}
```

Fig. 6. *LoR_IC* procedure

computing a new boundary formula, *PC_LoR* often has to go far back to tighten boundary formulas computed earlier. This tightening is done to make up for additional relaxation of transition relations of previous time frames. The great performance of IC3 suggests that tightening of boundary formulas of previous time frames can be efficiently done by adding inductive clauses. Second, IC3 builds an inductive invariant by tightening property P with inductive clauses. This may result in poor performance if an inductive invariant is "far away" from P. Sequential equivalence checking is an example of a PC problem where IC3 may perform poorly (see Section II). *LoR_IC* is meant to address this issue.

The pseudo-code of *LoR_IC* is shown in Figure 6. The lines where *LoR_IC* is different from *PC_LoR* are marked with an asterisk. Consider how *LoR_IC* builds formula H_j after formulas $H_0, \dots, H_{j-1}$ satisfying the four CO conditions have been generated. Similarly to *PC_LoR*, *LoR_IC* makes two steps to guarantee that the second CO condition i.e. $H_j \rightarrow P$ holds. In the first step, it makes sure that $H_{j-1} \wedge T^{rlx}_{j-1,j} \rightarrow P$ holds. However, in contrast to *PC_LoR*, this is done by calling procedure $RemBadSt_{IC}$ generating inductive clauses. If there is an H_{j-1}-state $\boldsymbol{s_{j-1}}$ that is one transition away from a bad state, a clause C inductive with respect to H_{j-1} is generated. This clause is falsified by $\boldsymbol{s_{j-1}}$ and so is added to H_{j-1} to exclude this state. The second step is performed like in *PC_LoR* by calling procedure $FinRlx$. The latter relaxes transition relation $T_{j-1,j}$ and builds H_j as a set set of clauses making up for this relaxation. This is where *LoR_IC* is *different* from IC3. In IC3, formula H_j is built by conjoining the inductive clauses generated to exclude H_{j-1}-states with P. Note that using these clauses when forming H_j is not actually mandatory. The "why-not" argument given in [1] is that these clauses are implied by $H_{j-1} \wedge T_{j-1,j}$.

To satisfy the third CO condition, *LoR_IC* calls function $ThirdCOcond_{IC}$. In contrast to $ThirdCOcond$ of *PC_LoR*, $ThirdCOcond_{IC}$ does the job by generation of inductive clauses. Suppose that one needs to eliminate an H_{j-1}-state $\boldsymbol{s_{j-1}}$ from which a state $\boldsymbol{s_j}$ falsifying H_j is reachable in one transition. Then $ThirdCOcond_{IC}$ generates a clause inductive with respect to H_{j-1}. This clause is falsified by $\boldsymbol{s_{j-1}}$ and so is added to H_{j-1} to exclude $\boldsymbol{s_{j-1}}$. To guarantee that the third and fourth CO conditions hold for all formulas H_k, $k = 0, \dots, j$ built so far, *LoR_IC* calls function $FinTouch_{IC}$. In contrast to $FinTouch$, $FinTouch_{IC}$ does the job via inductive clauses.

Note that instead of initializing H_j to 1 (line 5 of Fig. 6), one can call procedure $EducatGuessRlx$ to apply a transition relation relaxation tailored to a particular system/property (see Subsection VI-A). We believe that the version of *LoR_IC* where $EducatGuessRlx$ employs relaxation described in Section II is a promising algorithm for sequential equivalence checking. The idea here is as follows. First, $EducatGuessRlx$ generates formula H_j that is close to an inductive invariant. Then some fine-tuning of H_j is done by adding inductive clauses generated when computing boundary formulas H_m, $m > j$.

VII. Conclusions

We introduced a new framework for Property Checking (PC) based on a method called Logic Relaxation (LoR). The appeal of PC by LoR is that an inductive invariant is the result of comparison of the original and relaxed transition systems. So the complexity of PC can be significantly reduced if the relaxed system is close to the original one. A key part of the LoR method is a technique called partial quantifier elimination. So it is extremely important to keep improving the performance of algorithms implementing this technique.

Appendix

Lemma 1: Let EQ^X_0 denote $EQ(X^N_0, X^K_0)$. Let $T_{0,1} = T^N_{0,1} \wedge T^K_{0,1} \wedge EQ^X_0$ be the transition relation specifying miter M of two identical circuits N and K in terms of initial time frame variables. Let the relaxed transition $T^{rlx}_{0,1}$ for miter M be equal to $T^N_{0,1} \wedge T^K_{0,1}$ i.e. $T_{0,1} = EQ^X_0 \wedge T^{rlx}_{0,1}$. Let EQ^S_1 denote $EQ(S^N_1, S^K_1)$. Let $s^N_{j,1}$ denote j-th state variable of circuit N of time frame 1. Then

a) $I_0 \wedge T_{0,1} \rightarrow EQ^S_1$
b) $I_0 \wedge T^{rlx}_{0,1} \rightarrow (s^N_{j,1} \equiv s^K_{j,1})$ if the value of variable $s^N_{j,1}$ remains the same for every state reachable from an I^N_0-state in one transition.
c) $\exists W_0[I_0 \wedge EQ^X_0 \wedge T^{rlx}_{0,1}] \equiv EQ^S_1 \wedge \exists W_0[I_0 \wedge T^{rlx}_{0,1}]$

Proof:

Item a). Since N and K are identical and I_0 implies EQ^S_1, miter M reaches only states ($\boldsymbol{s_N}$,$\boldsymbol{s_M}$) where $\boldsymbol{s_N} = \boldsymbol{s_M}$. This means that $I_0 \wedge T_{0,1} \rightarrow EQ^S_1$.

Item b). This item explains under what conditions some clauses of EQ^X_0 are redundant without adding any clauses of EQ^S_1. Suppose that the assumption of item b) holds. Then the fact that $T^{rlx}_{0,1}$ does not impose restrictions on X^N and X^K does not matter as far as variables $s^N_{j,1}$ and $s^K_{j,1}$ are concerned. Indeed, the value of those variables remains the same for all assignments to X_N and X_K. This means that $I_0 \wedge T^{rlx}_{0,1} \rightarrow (s^N_{j,1} \equiv s^K_{j,1})$. So when taking formula EQ^X_0 out of the scope of quantifiers, adding all the clauses of EQ^S_1 is not necessary. (Recall that given sets of Boolean variables $A = (a_1, \dots, a_k)$

and $B = (b_1, \dots, b_k)$, $EQ(A,B) = (a_1 \equiv b_1) \wedge \dots \wedge (a_k \equiv b_k))$. Namely, one does not need to add the clauses of EQ^S_1 specifying $s^N_{j,1} \equiv s^K_{j,1}$.

Item c). Assume the contrary. Taking into account that $T_{0,1} = EQ^X_0 \wedge T^{rlx}_{0,1}$, this means that $\exists W_0[I_0 \wedge T_{0,1}] \not\equiv EQ^S_1 \wedge \exists W_0[I_0 \wedge T^{rlx}_{0,1}]$. Let *Left_part* and *Right_part* specify the left and right parts of the inequality above respectively. Consider the two alternatives.

Left_part = 1, *Right_part* = 0. Then there is an assignment $\boldsymbol{t}$ to $W_0 \cup S_1$ that satisfies $I_0 \wedge T_{0,1}$ and hence $I_0 \wedge T^{rlx}_{0,1}$. Since *Right_part* = 0, then $EQ^S_1(\boldsymbol{t}) = 0$, which means that $I_0 \wedge T_{0,1} \not\rightarrow EQ^S_1$. So we have a contradiction.

Left_part = 0, *Right_part* = 1 for an assignment $\boldsymbol{s_1}$ to S_1. Then there is an assignment $\boldsymbol{t}$ to $W_0 \cup S_1$ obtained by extending $\boldsymbol{s_1}$ that satisfies $EQ^S_1 \wedge I_0 \wedge T^{rlx}_{0,1}$. Since *Left_part* = 0, then $\boldsymbol{t}$ falsifies $I_0 \wedge T_{0,1}$. This means that $\boldsymbol{t}$ falsifies EQ^X_0 i.e. $\boldsymbol{x^N_0} \neq \boldsymbol{x^K_0}$ where $\boldsymbol{x^N_0}$ and $\boldsymbol{x^K_0}$ are assignments to X^N_0 and X^K_0 from $\boldsymbol{t}$. Let $\boldsymbol{t^*}$ be the assignment obtained from $\boldsymbol{t}$ by replacing assignment to $X^K_0 \cup Y^K_0$ specifying the execution trace for $\boldsymbol{x^K_0}$ with that specifying the execution trace for input $\boldsymbol{x^N_0}$. It is not hard to see that $\boldsymbol{t^*}$ has the same assignment to S_1 as $\boldsymbol{t}$ but satisfies $I_0 \wedge T_{0,1}$. So *Left_part* = 1 for assignment $\boldsymbol{s_1}$ to S_1 and we have a contradiction. ■

Proposition 1: Let H_j be a formula (depending only on variables of j-th cut) such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$. Then H_j is a boundary formula for the pair (ξ, ξ^{rlx}_j).

Proof: Assume the contrary i.e. H_j is not a boundary formula. Definition 5 suggests that then one of the two cases below takes place.

Case 1: There is a valid trace t=($\boldsymbol{s_0}$,...,$\boldsymbol{s_j}$) of ξ^{rlx}_j such that $\boldsymbol{s_j}$ is not reachable in ξ in j transitions and $H_j(\boldsymbol{s_j}) = 1$. Since t is a valid trace in ξ^{rlx} and $H_j(\boldsymbol{s_j}) = 1$, formula $H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$ evaluates to 1 under assignment $\boldsymbol{s_j}$ to S_j. Then $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j]$ evaluates to 1 under $\boldsymbol{s_j}$ as well, which means that $\boldsymbol{s_j}$ is reachable in ξ. So we have a contradiction.

Case 2: There is a valid trace t=($\boldsymbol{s_0}$,...,$\boldsymbol{s_j}$) of ξ and yet $H_j(\boldsymbol{s_j}) = 0$. Then formula $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j]$ evaluates to 1 under assignment $\boldsymbol{s_j}$. On the other hand, the fact that $H_j(\boldsymbol{s_j}) = 0$ means that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \neq H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$ under assignment $\boldsymbol{s_j}$. So we have a contradiction. ■

Proposition 2: Let $T_{j-1,j} = T^{rlx}_{j-1,j} \wedge R_{j-1,j}$. Let H_j be a formula such that $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j \wedge R_{j-1,j}] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$. Then H_j is a boundary formula for the pair (ξ, ξ^{rlx}_j).

Proof: By definition, $\mathbb{T}_j = \mathbb{T}^{rlx}_j \wedge R_{j-1,j}$. Then the correctness of the proposition follows from Proposition 1. ■

Proposition 3: $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \equiv \exists \mathbb{W}_{j-1}[\mathbb{H}_j \wedge \mathbb{T}^{RLX}_j]$.

Proof: Let us prove the proposition by induction. Proposition 1 entails that the proposition at hand holds for $j = 1$. Let us show that the correctness of the proposition for $j > 1$, implies that it holds for $j + 1$. Let ϕ denote $\exists \mathbb{W}_j[I_0 \wedge \mathbb{T}_{j+1}]$. Formula ϕ can be rewritten as $\exists W_j \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j \wedge T_{j,j+1}]$. Taking into account that $T_{j,j+1}$ does not depend on variables of $\mathbb{W}_{j-1}$, formula ϕ can represented as $\exists W_j[T_{j,j+1} \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j]]$. Using the inductive hypothesis this formula can be transformed into $\exists W_j[T_{j,j+1} \wedge \exists \mathbb{W}_{j-1}[\mathbb{H}_j \wedge \mathbb{T}^{RLX}_j]]$. Taking into account that $T_{j,j+1} = T^{rlx}_{j,j+1} \wedge R_{j,j+1}$, formula ϕ can be represented as $\exists \mathbb{W}_j[\mathbb{H}_j \wedge \mathbb{T}^{RLX}_j \wedge T^{rlx}_{j,j+1} \wedge R_{j,j+1}]$. Since H_{j+1} is obtained by taking $R_{j,j+1}$ out of the scope of quantifiers, formula ϕ can be rewritten as $H_{j+1} \wedge \exists \mathbb{W}_j[\mathbb{H}_j \wedge \mathbb{T}^{RLX}_j \wedge T^{rlx}_{j,j+1}]$. So the original formula ϕ is logically equivalent to formula $\exists \mathbb{W}_j[\mathbb{H}_{j+1} \wedge \mathbb{T}^{RLX}_{j+1}]$. ■

Proposition 4: Let $T_{j-1,j} = T^{rlx}_{j-1,j} \wedge R_{j-1,j}$, $j > 0$. Let formulas $H_0, \dots, H_j$ be built consecutively as follows. Formula H_0 equals I and formula H_j, $j > 0$ is built to satisfy $\exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_j \wedge R_{j-1,j}] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_j]$. Then $H_0, \dots, H_j$ are boundary formulas.

Proof: The fact that H_0 is a boundary formula follows from Definition 5. Let us show that formulas $H_1, \dots, H_j$ are also boundary by induction. Assume that formulas $H_1, \dots, H_{j-1}$ are boundary and show that then H_j is a boundary formula as well.

Proposition 3 entails that formula $\exists \mathbb{W}_{j-2}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_{j-1}]$ can be replaced with $\exists \mathbb{W}_{j-2}[I_0 \wedge \mathbb{T}_{j-1}]$. So $\exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_j \wedge R_{j-1,j}]$ can be rewritten as $\exists W_{j-1}[\exists \mathbb{W}_{j-2}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_{j-1} \wedge T^{rlx}_{j-1,j} \wedge R_{j-1,j}]]$, then as $\exists W_{j-1}[\exists \mathbb{W}_{j-2}[I_0 \wedge \mathbb{T}_{j-1} \wedge T^{rlx}_{j-1,j} \wedge R_{j-1,j}]]$ and finally as $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j]$. Similarly formula $H_j \wedge \exists \mathbb{W}_{j-1}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_j]$ can be rewritten as $H_j \wedge \exists W_{j-1}[\exists \mathbb{W}_{j-2}[\mathbb{H}_{j-1} \wedge \mathbb{T}^{RLX}_{j-1} \wedge T^{rlx}_{j-1,j}]]$, then as $H_j \wedge \exists W_{j-1}[\exists \mathbb{W}_{j-2}[I_0 \wedge \mathbb{T}_{j-1} \wedge T^{rlx}_{j-1,j}]]$ and finally as $H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$. So H_j satisfies $\exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}_j] \equiv H_j \wedge \exists \mathbb{W}_{j-1}[I_0 \wedge \mathbb{T}^{rlx}_j]$. Then from Proposition 1 it follows that H_j is a boundary formula. ■

Proposition 5: Let H_j, $j = 1, \dots, m$ be formulas derived by *PC_LoR* for m time frames where $H_j \rightarrow P$. Then property P holds for system ξ for at least m transitions.

Proof: As we mentioned in Subsection V-C, $I_0 \wedge \mathbb{T}_j \rightarrow H_j$ holds. Then $H_j \rightarrow P$ entails $I_0 \wedge \mathbb{T}_j \rightarrow P$. ■

Proposition 6: *PC_LoR* is sound.

Proof: Consider the two obvious alternatives.

The answer is "property fails". This answer is returned by *PC_LoR* if there is an assignment $\boldsymbol{t}$ satisfying $I_0 \wedge \mathbb{H}_j \wedge \mathbb{T}^{RLX}_j$ and $\overline{P}$. From Proposition 3 it follows, that then there is an assignment $\boldsymbol{t^*}$ satisfying $I_0 \wedge \mathbb{T}_j$ and $\overline{P}$. Hence there is a counterexample of length $j + 1$.

The answer is "property holds". This answer is returned when there appear a formula H_{j-1} such that $H_j \rightarrow H_{j-1}$ and $H_{j-1} \wedge T^{rlx}_{j-1,j} \rightarrow H_j$ hold. Since H_{j-1} implies H_j, then $H_{j-1} \equiv H_j$. Since $T_{j-1,j}$ implies $T^{rlx}_{j-1,j}$, $H_{j-1} \wedge T_{j-1,j} \rightarrow H_j$ holds as well and H_{j-1} is an inductive invariant. ■

Proposition 7: *PC_LoR* is complete.

Proof: Consider the following alternatives.

Property P fails. Let j be the first time frame where a bad state $\boldsymbol{s_j}$ is reachable by ξ. Let H_j be a boundary formula generated for j-th time frame. Since H_j is implied by $I_0 \wedge \mathbb{T}_j$, *PC_LoR* will not be able to make H_j imply P. Then procedure $RemBadSt$ will terminate reporting that P failed.

Property P holds. Let $H_0, \ldots, H_m$, be a sequence of boundary formulas built by *PC_LoR* . Let $H_{j-1} \rightarrow H_j$ hold for every j, $0 < j \leq m$. If $m > 2^{|S|}$ where S is the set of state variables, there has to be a formula H_{j-1} that is logically equivalent to H_j. Since $H_{j-1} \wedge T^{rlx}_{j-1,j} \rightarrow H_j$ holds, *PC_LoR* will terminate reporting that P holds. ■

References

[1] A. R. Bradley. Sat-based model checking without unrolling. In *VMCAI*, pages 70–87, 2011.

[2] E. Goldberg. Boundary points and resolution. In *Proc. of SAT*, pages 147–160. Springer-Verlag, 2009.

[3] E. Goldberg. Equivalence checking and simulation by computing range reduction. Technical Report arXiv:1507.02297 [cs.LO], 2015.

[4] E. Goldberg. Equivalence checking by logic relaxation. Technical Report arXiv:1511.01368 [cs.LO], 2015.

[5] E. Goldberg and P. Manolios. Quantifier elimination by dependency sequents. In *FMCAD-12*, pages 34–44, 2012.

[6] E. Goldberg and P. Manolios. Quantifier elimination via clause redundancy. In *FMCAD-13*, pages 85–92, 2013.

[7] E. Goldberg and P. Manolios. Bug hunting by computing range reduction. Technical Report arXiv:1408.7039 [cs.LO], 2014.

[8] E. Goldberg and P. Manolios. Partial quantifier elimination. In *Proc. of HVC-14*, pages 148–164. Springer-Verlag, 2014.

[9] E. Goldberg and P. Manolios. Quantifier elimination by dependency sequents. *Formal Methods in System Design*, 45(2):111–143, 2014.

[10] K. L. Mcmillan. Interpolation and sat-based model checking. In *CAV-03*, pages 1–13. Springer, 2003.