arXiv:1408.7039v4 [cs.LO] 11 Oct 2014

Bug Hunting By Computing Range Reduction

Eugene Goldberg and Panagiotis Manolios
College of Computer and Information Science
Northeastern University, USA, 360 Huntington Ave., Boston MA 02115, USA
email: {eigold,pete}@ccs.neu.edu

***Abstract*—We describe a method of model checking called Computing Range Reduction (CRR). The CRR method is based on derivation of clauses that reduce the set of traces of reachable states in such a way that at least one counterexample remains (if any). These clauses are derived by a technique called Partial Quantifier Elimination (PQE). Given a number n, the CRR method finds a counterexample of length less or equal to n or proves that such a counterexample does not exist[1]. We show experimentally that a PQE-solver we developed earlier can be efficiently applied to derivation of constraining clauses for transition relations of realistic benchmarks.**

One of the most appealing features of the CRR method is that it can potentially find long counterexamples. This is the area where it can beat model checkers computing reachable states (or their approximations as in IC3) or SAT-based methods of bounded model checking. PQE cannot be efficiently simulated by a SAT-solver. This is important because the current research in model checking is dominated by SAT-based algorithms. The CRR method is a reminder that one should not put all eggs in one basket.

I. Introduction

In this paper, we introduce a new method of model checking called Computing Range Reduction (CRR).

A. Motivating example

Let ξ be a state transition system. Let formula I specify the initial states of ξ and formula P specify a property that does not hold for ξ. Suppose that we know that there exists *only one* sequence $D = (\boldsymbol{s_0},\dots,\boldsymbol{s_n})$ of states from an initial state to a bad state. Suppose that we know only the state $\boldsymbol{s_0}$ of this sequence (that is an initial state of ξ) and want to find the remaining states $\boldsymbol{s_1},\dots,\boldsymbol{s_n}$. We assume here that a **counterexample** E is a trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_{n-1}},\boldsymbol{x_{n-1}})$ where $\boldsymbol{x_i}$ is a complete assignment to combinational input variables in i-th time frame. As usual, we assume that ξ transitions to state $\boldsymbol{s_{i+1}}$ from $\boldsymbol{s_i}$ under assignment $\boldsymbol{x_i}$, $i = 0,\dots,n-1$. So trace E leads to state $\boldsymbol{s_n}$. A traditional model checker cannot exploit the fact that every counterexample E goes through the same sequence of states D. To find E, such a model checker would have to build a sequence of sets of states $A_1,\dots,A_n$ where A_i is the set of states *reachable* from $\boldsymbol{s_0}$ in i transitions or an over-approximation thereof. For the sake of simplicity, below, we assume that A_i is the precise set of states reachable from $\boldsymbol{s_0}$ in i transitions.

[1]To make exposition simpler, in this paper, we formulate a version of the CRR method that proves only that a property holds for n transitions. However, the CRR method can be modified to become complete and hence able to prove that a property holds for an arbitrary number of transitions. We are planning to publish this modification of the CRR method in the near future.

In reality, finding a counterexample E does not require computing sets A_i,$i = 1,\dots,n$. Let B_i denote the set of states that are reachable in i transitions from initial states *different* from $\boldsymbol{s_0}$. (We assume here that I specifies more than one initial state.) Every state $\boldsymbol{s_i}$ of D is in $A_i \setminus B_i$, $i = 1,\dots,n$. Indeed, $\boldsymbol{s_i}$ cannot be in B_i because then D would not be the only sequence of states leading to a bad state. Importantly, the size of the set $A_i \setminus B_i$ can be dramatically smaller than A_i.

The CRR method is able to find a counterexample E by generating only sets of states $A_i \setminus B_i$. Here is how it is done. Instead of finding the set of states reachable from state $\boldsymbol{s_0}$ in i transitions, the CRR method builds the set of states that become *unreachable* in i transitions if the state $\boldsymbol{s_0}$ is *excluded* from the set of initial states. It is not hard to see that this is exactly the set $A_i \setminus B_i$ because the latter consists of states that are reachable in i transitions *only* from state $\boldsymbol{s_0}$. Obviously, these states become unreachable if $\boldsymbol{s_0}$ is excluded. The fact that set $A_n \setminus B_n$ contains a *bad* state $\boldsymbol{s_n}$ means that $\boldsymbol{s_n}$ is reachable from $\boldsymbol{s_0}$ in n transitions. Hence property P fails.

B. Operation of the CRR method in more detail

Let us use the example above to describe the operation of the CRR method in more detail. Let N be a circuit specifying the transition relation of system ξ. Let S and X be the sets of state variables and combinational input variables of N respectively. So $S \cup X$ is the set of input variables of N. The key operation of the CRR method is to **exclude** some input assignments of the initial time frame and compute the set of reachable states that become unreachable due such an exclusion.

In our example, the set of excluded inputs is specified by clause C that is falsified only by state $\boldsymbol{s_0}$. This clause excludes every assignment $(\boldsymbol{s_0},\boldsymbol{x})$ where $\boldsymbol{x}$ is an arbitrary complete assignment to X. To compute the effect of constraining inputs of the initial time frame by C, a set of **range reduction formulas** $H_1,\dots,H_n$ is constructed. (We assume that time frames are indexed starting with 0. So the initial time frame has index 0.) Formula H_i evaluates to 0 for state $\boldsymbol{s}$ iff $\boldsymbol{s}$ is reachable in i-transitions but becomes unreachable in i-transitions after removing the traces excluded by C. In our example, clause C excludes every trace that starts with state $\boldsymbol{s_0}$. So, the set of states falsifying formula H_i is equal to $A_i \setminus B_i$.

The name "range reduction formula" is due to the fact that H_i specifies the reduction of the range of a combinational circuit caused by excluding its inputs by C. This circuit is a composition of i copies N. Formulas H_i are computed one by one. Once formula H_i is formed, the CRR method checks if $\overline{H_i} \rightarrow P$ holds. If it does not, then there is a bad state $\boldsymbol{s_i}$ that becomes unreachable in i transitions after excluding state $\boldsymbol{s_0}$. Hence $\boldsymbol{s_i}$ is reachable from $\boldsymbol{s_0}$ and P does not hold. If $\overline{H_i} \rightarrow P$ holds, then the CRR method computes the next range reduction formula H_{i+1}. This goes on until a bad state falsifying the most recent range reduction formula is found.

So far, we assumed that there exists only one sequence of states D from an initial state to a bad state and this sequence specifies counterexamples of length n. Suppose that this is not the case. That is either property P holds for n transitions or for every i less or equal to n, clause C does not exclude all counterexamples of length i (if any). Then, $\overline{H_i} \rightarrow P$ holds for every range reduction formula H_i. This means that excluding the inputs of the initial time frame falsified by clause C does not affect the answer to the question whether P holds for n transitions. In this case, we will say that clause C is **$\boldsymbol{P^n}$-equivalent**.

Given a number n, the CRR method either finds a counterexample of length at most n or proves that P holds for n transitions. The latter is done by generating P^n-equivalent clauses until one of the two conditions below is met.

1) The set of all possible traces of n transitions reduces to one trace consisting of only good states.
2) The set of all possible traces of m transitions where $m \leq n$ reduces to one trace L where
   - all states of L are good and
   - the last state of L repeats some previous state of L

C. What sets CRR method apart from competition

One of the most appealing features of the CRR method is that it can potentially detect **very deep bugs**. Such bugs are hard to find by the existing methods. The complete methods based on computing reachable states or their over-approximation work in a *breadth-first* manner. That is they consider counterexamples of length n only *after* they proved that no counterexample of length $n-1$ or less exists. This also applies to Bounded Model Checking (BMC). The breadth-first search strategy makes these methods very inefficient in finding deep bugs. As we mentioned above, when the CRR method looks for a counterexample, it generates range reduction formulas $H_1, \dots H_n$. This means that the CRR method looks for counterexamples in a *depth-first* manner. In particular, the CRR-method can find a counterexample of length n without proving that counterexamples of length less than n do not exist. This can be done efficiently because the CRR method computes only a small subset of the set of states reachable in i transitions $i = 1, \dots, n$.

D. Partial quantifier elimination

Computing a range reduction formula H_i comes down to solving an instance of the Partial Quantifier Elimination (PQE) problem [2], [3]. In general, a PQE-solver cannot be efficiently simulated by a SAT-solver. This is important because the current research in model checking is dominated by SAT-based approaches. The CRR method is a reminder that one should not put all eggs in one basket.

In the experimental part of the paper, we give some results of applying our PQE-algorithm [2] to constructing range reduction formulas. We compute such formulas for transition relations of the HWMCC-10 benchmarks. Our experiments show that even the current version of the PQE algorithm that has huge room for improvement can be successfully applied to computing range reduction formulas.

E. Structure of the paper

This paper is structured as follows. In Section II, we present a simple example illustrating the operation of the CRR method. We also discuss the advantages of the CRR method in finding deep bugs. Section III gives a brief introduction into partial quantifier elimination. Basic definitions are given in Section IV. In Section V, we explain the main idea of the CRR-method. Section VI introduces the important classification of traces as isolated or public with respect to a constraining clause. Application of CRR to bug hunting is discussed in Section VII. In Section VIII, we explain how the CRR method identifies P^n-equivalent clauses. We compare the CRR method with other model checkers in Section IX. Section X describes a model checker called MC_CRR that is based on the CRR method. Experimental results are given in Section XI. In Section XII, we make some conclusions.

II. An Example Of How CRR Method Operates

In this section, we describe the operation of the CRR method when checking a property of an *abstract* k-bit counter. An abstract counter is a regular counter where no assumptions about the binary encodings of numbers are made. In particular, a pair of consecutive numbers can have completely different binary representations.

One can view an abstract counter as describing a sub-behavior of a sequential circuit going through a long sequence of states K where all states of K are unique. The counter has a combinational variable x whose value specifies whether this counter stays in the current state or moves to the next state of K. Since an abstract counter is meant to simulate a long sequence of unique states of an arbitrary sequential circuit, it is reasonable to avoid making any assumptions about the way states are encoded.

Subsection II-A describes the example with an abstract counter in more detail. Application of the CRR method to this example is described in Subsection II-B. Subsection II-C uses an abstract counter to show the advantage of the CRR method over existing methods in finding deep bugs.

A. Problem description

An abstract k-bit counter is specified by a sequential circuit ξ defined as follows. Let $S = \{s_1, \dots, s_k\}$ specify the set of state variables of ξ and x be the only combinational input variable of ξ. We will assume that ξ has only one initial state where $s_i = 0, i = 1, \dots, k$. We will denote the initial state as $\boldsymbol{s_{init}}$. Let $Val(\boldsymbol{s})$ denote the number stored by the counter in state $\boldsymbol{s}$. As we mentioned above, we do not assume any relation between $Val(\boldsymbol{s})$ and $\boldsymbol{s}$. Our only constraints are that $\boldsymbol{s} \neq \boldsymbol{s^*}$ implies $Val(\boldsymbol{s}) \neq Val(\boldsymbol{s^*})$ and that $Val(\boldsymbol{s_{init}}) = 0$.

The transition relation of ξ is specified as follows. Let $\boldsymbol{s}$ be the current state of ξ.

- If $x = 0$, ξ remains in state $\boldsymbol{s}$.
- If $x = 1$,
  - if $Val(\boldsymbol{s}) \neq 2^k - 1$, ξ switches to state $\boldsymbol{s'}$ such that $Val(\boldsymbol{s'}) := Val(\boldsymbol{s}) + 1$.
  - if $Val(\boldsymbol{s}) = 2^k - 1$, ξ resets i.e. switches to $\boldsymbol{s_{init}}$.

Let $P(S)$ be a formula such that $P(\boldsymbol{s}) = 1$ iff $Val(\boldsymbol{s}) < d$. The problem we want to solve is to check if ξ satisfies property P for n transitions. To prevent resetting the counter, we will assume that $n < 2^k - 1$. Since $Val(\boldsymbol{s_{init}})$=0 and one transition increases the value corresponding to the current state by at most 1, P holds if $n < d$ and fails otherwise.

B. Application of the CRR method

Here is how the problem above is solved by the CRR method. Let $C = s_1 \vee \cdots \vee s_k \vee \overline{x}$ be a clause selected by the CRR method to constrain input assignments of the initial time frame. Namely, C removes every input assignment in which $s_i = 0, i = 1, \ldots, k$ and $x = 1$. Let $H_1, H_2, \ldots,$ be range reduction formulas computed with respect to clause C. We will say that H_i **excludes** state $\boldsymbol{s}$ from i-th time frame if $H_i(\boldsymbol{s}) = 0$.

If inputs of the initial time frame are not constrained by C, the set of states of the counter reachable in i transitions consists of the $i+1$ states with values ranging from 0 to i. If E is a trace of i transitions and x is equal to 1 in m time frames and equal to 0 in $i - m$ time frames, the counter reaches a state $\boldsymbol{s}$ where $Val(\boldsymbol{s}) = m$. If inputs of the initial time frame are constrained by C, variable x cannot have value 1 in the initial time frame. So traces where x is equal to 1 in *every* time frame are excluded. This means that the state $\boldsymbol{s}$ where $Val(\boldsymbol{s})$=i is excluded by H_i. Note that every state $\boldsymbol{s}$ such that $0 \leq Val(\boldsymbol{s}) < i$ is reachable in i transitions by a trace where $x = 0$ in the initial time frame i.e. by an *allowed* trace. Hence such a state cannot be excluded by H_i and so $H_i(\boldsymbol{s}) = 1$.

Suppose that $n \geq d$. For every new range reduction formula H_i, the CRR method checks if $\overline{H_i} \rightarrow P$ holds. For the first $d-1$ formulas $H_1, \ldots, H_{d-1}$, this implication holds and so no bad state is excluded. However, since H_d excludes a state $\boldsymbol{s}$ such that $Val(\boldsymbol{s}) = d$ and hence $P(\boldsymbol{s})$=0, implication $\overline{H_d} \rightarrow P$ does not hold. At this point, the CRR method reports that P is broken by a trace of d transitions.

Now, assume that $n < d$. Then $\overline{H_i} \rightarrow P$ holds for all formulas $H_1, \ldots, H_n$. This means that formula C is P^n-equivalent. That is constraining the inputs of the initial time frame of ξ with C does not affect the answer to the question whether P holds for n transitions. In general, one needs to add many P^n-equivalent clauses to prove that a property holds for system ξ for n transitions. However, for our example, showing that C is P^n equivalent is sufficient to finish the job. Note that only state $\boldsymbol{s_{init}}$ is possible in the initial time frame. Due to clause C, the value of x in the initial time frame is fixed at 0. So only state $\boldsymbol{s_{init}}$ is possible in the next time frame that is the same state as in the previous time frame.

At this point the CRR method stops to declare that P holds for n transitions. In Subsection I-B, we gave two conditions under which the CRR method claims that a property holds for n transitions. Our example employs the second condition. The set of all traces of m transitions where $m \leq n$ reduces to one trace L where the last state repeats a previously seen state of L. In our example, L consists of two copies of state $\boldsymbol{s_{init}}$ and m is equal to 1.

C. Comparison of the CRR method with other model checkers

In this subsection, we use our example to discuss the advantage of the CRR method over other model checkers in the context of bug hunting. To be concrete, let us assume that d=20,000 and one needs to check if the property P above holds for some n. We will assume that $n > d$ and so P does not hold.

To find a counterexample by BMC, one will have to generate formulas $G_1, \ldots, G_{20000}$ where satisfiability of G_i means the existence of a counterexample of i transitions. Formula G_i contains i copies of the transition relation. So even if ξ is small, formulas G_i grow too large to be solved efficiently by a SAT-algorithm.

A model checker computing the set of states reachable in k transitions $k = 1, \ldots, n$ or its over-approximation will have a different kind of a problem. Before searching for a counterexample of 20,000 transitions, such a model checker will have to prove that no counterexample of at most 19,999 transitions exists. This requires computing 19,999 sets of reachable states or their over-approximations.

The computation above can be done efficiently only for particular binary encodings of the values of the counter. Consider, for instance, the usual binary encoding where the more significant a state bit is the less frequently it toggles when the counter switches from the current state to the next one. In this case, there is a natural ordering of state variables for which the set of states of the counter reachable in k transitions can be represented by a compact BDD. So a BDD- based model checker will have no problem with finding a counterexample.

An IC3-like model checker that builds over-approximations of the set of reachable states will also benefit of the encoding above. A key operation of IC3 is to compute an inductive clause. To make this computation efficient, state encoding should satisfy the following property. If there is a transition from state $\boldsymbol{s}$ to state $\boldsymbol{s'}$, the Hamming distance between $\boldsymbol{s}$ and $\boldsymbol{s'}$ should be small. In the majority of transitions, the encoding above satisfies this property. So, most likely, an IC3-like model checker will find a counterexample efficiently.

As we mentioned above an abstract counter is meant to simulate a sub-behavior of a sequential circuit, so, in general, no assumptions about state encoding can be made. In this case, the size of a BDD representing the set of states reachable in k transitions can be large no matter how variables are ordered. So finding a counterexample by a BDD based model checker becomes inefficient. The same applies to an IC3-like model checker. The reason is that generation of inductive clauses becomes inefficient.

As we showed above, in our example, the CRR method builds range reduction formulas H_i, $i = 1, \ldots, k$ that exclude only *one* state from i-th time frame. That is to reach a bad state, the CRR method needs to compute only one state per time frame as opposed to computing the set of *all* states reachable in i transitions or its over-approximation. For that reason, for our example, the CRR method has very weak dependence on state encoding (if any). So, arguably, it will be able to find a counterexample in cases where other model checkers will fail.

III. Partial Quantifier Elimination

In this section, we recall Quantifier Elimination (QE) and Partial QE (PQE) the latter being a key operation of the CRR method. This section is structured as follows. Subsection III-A defines the QE and PQE problems. We introduce the notion of a noise-free PQE-solver in Subsection III-B. This notion plays an important role in reasoning about range reduction formulas that we introduce in Section V. In Subsection III-C, we show that computing the range of a circuit or reduction of the circuit range caused by input constraints come down to QE and PQE respectively.

A. Quantifier elimination and partial quantifier elimination

Let $G(X,Y)$ be a CNF formula. We will call formula $\exists X[G]$ an $\exists$***CNF***. The problem of **Quantifier Elimination** (**QE**) is to find a quantifier-free formula $H(Y)$ such that $H \equiv \exists X[G]$.

Let $\exists X[F(X,Y) \wedge G(X,Y)]$ be an $\exists$CNF. The problem of **Partial QE** (**PQE**) is to find a quantifier-free formula $F^*(Y)$ such that $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$. We will say that formula F^* is obtained by taking F out of the scope of quantifiers in $\exists X[F \wedge G]$.

An obvious difference between PQE from QE is that the latter takes *the entire formula* $F \wedge G$ out of the scope of quantifiers. Importantly, PQE can be dramatically simpler than QE especially if formula F is much simpler than G. In Section VII we show that computing range reduction formulas comes down to an instance of the PQE problem. In this instance, PQE is drastically simpler than QE because only a small part of the formula is taken out of the scope of quantifiers.

B. Noise-free PQE-solver

Let $F^*(Y)$ be a solution to the PQE problem i.e. $F^* \wedge \exists X[G] \equiv \exists X[F \wedge G]$. Recall that Y denotes the set of free variables of $\exists X[F \wedge G]$. Let C be a clause of F^* that is implied by G. Then formula $F^* \setminus \{C\}$ is also a solution to the same PQE problem. That is $F^{**} \wedge \exists X[G] \equiv \exists X[F \wedge G]$ where $F^{**} = F \setminus \{C\}$. One can think of clauses of F^* implied by G as "noise".

Suppose that a clause C of F^* is not implied by G but by adding literals of variables from $Y \setminus \mathit{Vars}(C)$ clause C can be extended to a clause implied by G. This can also be viewed as the presence of some noise in C. We will say that a clause of F^* is **noise-free** if the extension above does not exist. We will call F^* **a noise-free solution** if every clause of F^* is noise-free. We will call a PQE algorithm **noise-free** if it produces only noise-free solutions. A clause C, a solution F^* and a PQE-algorithm that are not noise-free are called **noisy**.

C. Relation of QE and PQE to computing range of a circuit

Let $M(X,Y,Z)$ be a multi-output combinational circuit where X,Y and Z specify input, intermediate and output variables of M respectively. In this subsection, we discuss QE and PQE in the context of computing the range of M. Namely, we show that a) computing the range of M comes down to QE; b) PQE can be used to compute range reduction caused by constraining inputs of M.

In the two propositions below, we assume that $G(X,Y,Z)$ is a CNF formula specifying circuit M that is obtained by Tseitsin transformations.

Proposition 1: Let $R(Z)$ be a CNF formula such that $R \equiv \exists W[G]$ where $W = X \cup Y$. (That is R is a solution to the QE problem.) Then the assignments satisfying $R(z)$ specify the range of M.

The proofs of all propositions are given in the appendix.

Proposition 2: Let $C(X)$ be a clause depending only on input variables of M. Let $H(Z)$ be a CNF formula such that $H \wedge \exists W[G] \equiv \exists W[C \wedge G]$ where $W = X \cup Y$. (That is H is a solution to the PQE problem.) Let H and H^* be a noise-free and noisy solution respectively. Then

1) The assignments *falsifying* H specify the range reduction in M caused by excluding inputs falsifying C. That is $H(\boldsymbol{z}) = 0$ iff
   - there is an input $\boldsymbol{x}$ for which circuit M produces output $\boldsymbol{z}$
   - all inputs for which M produces output $\boldsymbol{z}$ falsify C
2) $H^* \rightarrow H$

IV. Notation and definitions

Let ξ be a state transition system with a transition relation specified by a combinational circuit $N(S,X,Y,S')$. Here S and S' are sets of present and next state variables, X is the set of combinational input variables, and Y is the set of internal combinational variables. Then $S \cup X$ (respectively S') specify the input variables (respectively output variables) of N. Let $T(S,X,Y,S')$ be a formula specifying N. Let $P(S)$ be a property of ξ and $I(S)$ be a formula specifying the set of initial states of ξ. For the sake of simplicity, in the following exposition we *omit mentioning the variables of* Y.

Definition 1: A complete assignment ($\boldsymbol{s}$,$\boldsymbol{x}$) to variables of (S,X) is called an **input pair**. We will refer to $\boldsymbol{s}$ (respectively $\boldsymbol{x}$) as a **state** (respectively $\boldsymbol{X}$**-input**). A sequence $(\boldsymbol{s_0},\boldsymbol{x_0}),\ldots,(\boldsymbol{s_k},\boldsymbol{x_k})$ of input pairs is called a **trace** of ξ if T($\boldsymbol{s_i}$,$\boldsymbol{x}$,$\boldsymbol{s_{i+1}}$)=1, $0 \leq i < k$. If $I(\boldsymbol{s_0}) = 1$, this trace is called **initialized**.

Definition 2: Let $E = (\boldsymbol{s_0},\boldsymbol{x_0}),\ldots,(\boldsymbol{s_k},\boldsymbol{x_k})$ be a trace. Let $\boldsymbol{s_{k+1}}$ be the state to which ξ transitions under input pair

$(\boldsymbol{s_k},\boldsymbol{x_k})$. We will call $\boldsymbol{s_{k+1}}$ **the state reachable by trace $\boldsymbol{E}$**. We will also say that $\boldsymbol{s_{k+1}}$ is reachable in $k+1$ transitions.

Definition 3: Given a property $P(S)$ of system ξ, a state $\boldsymbol{s}$ is called **good** (respectively **bad**) if $P(\boldsymbol{s}) = 1$ (respectively $P(\boldsymbol{s}) = 0$). Property P is false for ξ if there is an initialized trace $E = (\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_k},\boldsymbol{x_k})$ such that

- every state $\boldsymbol{s_i}$ of E is good $i = 0,\dots,k$
- the state $\boldsymbol{s_{k+1}}$ reachable by E is bad

Trace E is called a **counterexample**.

Definition 4: We will index variables of system ξ to distinguish between different time frames. We will assume that numbering of time frames starts with 0. We will refer to the time frame with index 0 as or the **initial time frame**.

Definition 5: Let $H(S,X)$ be a CNF formula that constrains the input pairs of the system ξ in the initial time frame. That is H excludes every initialized trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_k},\boldsymbol{x_k}),\dots$ in which $(\boldsymbol{s_0},\boldsymbol{x_0})$ falsifies H. We will refer to such traces as **excluded by formula $\boldsymbol{H}$**. If the input pair $(\boldsymbol{s_0},\boldsymbol{x_0})$ of an initialized trace satisfies H, this trace is said to be **allowed** by H.

Definition 6: Let P be a property of system ξ. Let $C(S,X)$ be a clause excluding input pairs of the initial time frame. Suppose that P holds for system ξ for n transitions iff the set of traces *allowed* by clause C contains a counterexample of length at most n. We will say that the system constrained by C is **$\boldsymbol{P^n}$-equivalent** to the original system ξ. Informally, P^n-equivalence means that discarding the traces of ξ excluded by C does not eliminate *all* counterexamples of length at most n (if any). We will call clause C preserving P^n-equivalence of system ξ a **$\boldsymbol{P^n}$-equivalent clause**.

V. Model Checking By CRR

In this section, we give an introduction into model checking by Computing Range Reduction (CRR). First, we outline the main idea in Subsection V-A. Then, in Subsection V-B, we give a high-level description of a model checker based on CRR.

A. Main idea

Let ξ be a system introduced in Section IV and P be a property of ξ. We will assume that $I \rightarrow P$, that is all initial states satisfy property P. Let $C(S,X)$ be a clause specified in terms of input variables of circuit N above such that $\overline{C} \rightarrow I$. Suppose that we use C to exclude traces as described in Definition 5. Suppose that a state $\boldsymbol{s}$ of ξ is reachable in i transitions only by traces excluded by C. This means that if one discards the traces excluded by C, state $\boldsymbol{s}$ becomes unreachable in i transitions. Such states are specified by range reduction formulas defined below.

Definition 7: The result of using clause C to exclude traces of ξ of length at most n can be characterized by a set of formulas $H_1,\dots,H_n$ defined as follows. The value of $H_i(\boldsymbol{s})$ is equal to 0 iff

- $\boldsymbol{s}$ is reachable in i transitions
- all traces of length i that reach $\boldsymbol{s}$ are excluded by C.

We will call H_i a **range reduction formula**. We will say that state $\boldsymbol{s}$ **is excluded by $\boldsymbol{H_i}$** if $H_i(\boldsymbol{s}) = 0$.

Model checking by Computing Range Reduction (CRR) is based on the following four observations. The first observation is that formula H_i specifies a *reduction in the range* of a circuit obtained by the composition of i circuits N. Such a circuit describes the traces of i transitions. The change of range described by H_i is caused by discarding traces excluded by clause C. Using Proposition 2, one can compute such range reductions by a PQE solver.

The second observation is that one can use range reduction formulas to find a counterexample. Suppose that $\overline{H_i} \not\rightarrow P$ i.e. there is a state $\boldsymbol{s}$ such that $H_i(\boldsymbol{s}) = 0$ and $P(\boldsymbol{s}) = 0$. This means that by discarding the traces excluded by clause C, one excludes a *bad* state $\boldsymbol{s}$ from the set of states reachable in i transitions. This implies that there is a counterexample formed by a trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_i},\boldsymbol{x_i})$ excluded by C leading to a bad state. This trace can be easily recovered from $H_1,\dots,H_i$ and clause C by $i+1$ SAT-checks. In more detail, bug hunting by CRR is described in Section VII.

The third observation is as follows. As mentioned in the introduction, formula H_i specifies the difference between sets A_i and B_i. Set A_i consists of the states that can be reached by traces of length i that are excluded by C. Set B_i is a subset of A_i that consists of the states that are also reachable by traces of length i that are *allowed* by C. The set $A_i \setminus B_i$ represented by H_i consists of the states that can be reached *only* by traces of length i excluded by C. This set can be *very small* even when sets A_i and B_i are huge. In Section XI, we give some experimental evidence to support this conjecture. Informally, this means that a model checker based on CRR can find a bug by examining a very small number of states.

The fourth observation is that one may not need to compute all n range reduction formulas H_i to prove that clause C is P^n-equivalent. Suppose, for example, that formula H_i is empty where $i < n$. That is $H_i \equiv 1$ (and hence H_i cannot exclude a bad state). Then every formula H_j, $i < j \leq n$ is also empty.

B. High-level description of a model checker based on CRR

In this subsection, we give a high-level explanation of how one can build a model checker based on CRR that checks if a property P holds for n transitions. A detailed description of an instance of such a model checker is given in Section X.

Definition 8: Let $(\boldsymbol{s},\boldsymbol{x})$ be an input pair where $\boldsymbol{s}$ is an initial state. Then we will call this pair an **initial input pair**.

Suppose that one excludes the initial input pairs of ξ as follows. First, an initial input pair $(\boldsymbol{s},\boldsymbol{x})$ is picked. Then a clause $C(S,X)$ falsified by $(\boldsymbol{s},\boldsymbol{x})$ is generated such that $\overline{C} \rightarrow I$. After that, range reduction formulas $H_1,\dots,H_n$ are computed with respect to clause C. If a formula H_i does not imply P i.e. H_i excludes a bad state, a counterexample is generated. Otherwise, one proves that C is a P^n-equivalent clause. After that, C is added to a formula Q that accumulates all P^n-equivalent clauses generated so far to exclude initial input pairs. Initially, Q is empty.

Then one picks an initial input pair ($\boldsymbol{s}$,$\boldsymbol{x}$) that satisfies Q. This guarantees that this a new initial input pair. A new clause C falsified by this input pair is generated and a new set of range reduction formulas is generated with respect to clause C. The process of elimination of initial input pairs goes on until either a counterexample is generated or all initial input pairs but one are excluded. The reason why the last initial input pair is not excluded is as follows. In Subsection V-A we mentioned that H_i represents the difference of sets A_i and B_i where B_i is a subset of A_i. The larger the set B_i, the smaller the set $A_i \setminus B_i$ that H_i represents. The size of the set B_i depends on the number of traces of length i that are *allowed* by clause C. If the last initial input pair is eliminated by a clause C, then *no trace of length* i is allowed by C. In this case, the set B_i is empty and the CRR method essentially reduces to reachability analysis where set A_i grows uncontrollably.

Let ($\boldsymbol{e_0}$,$\boldsymbol{d_0}$) be the initial input pair that still satisfies Q. This means that every remaining counterexample (if any) starts with the input pair ($\boldsymbol{e_0}$,$\boldsymbol{d_0}$). Let $\boldsymbol{e_1}$ denote the state to which ξ transitions to under input ($\boldsymbol{e_0}$,$\boldsymbol{d_0}$). Obviously, the traces of ξ allowed by Q go through state $\boldsymbol{e_1}$. This means that the original system ξ with initial states specified by formula I is P^n-equivalent with ξ that has only one initial state equal to $\boldsymbol{e_1}$. This also means that the initial time frame can be discarded.

One can use the same procedure of building formula Q that excludes the initial input pairs of the modified ξ. This initial input pairs are of the form ($\boldsymbol{e_1}$,$\boldsymbol{x}$) where $\boldsymbol{x}$ is a complete assignment to variables of X. The procedure described above can be used to eliminate all initial input pairs but an input pair ($\boldsymbol{e_1}$,$\boldsymbol{d_1}$). This means that every remaining counterexample of the original system ξ has to start with ($\boldsymbol{e_0}$,$\boldsymbol{d_0}$),($\boldsymbol{e_1}$,$\boldsymbol{d_1}$).

The procedure of elimination of initial input pairs has the following three outcomes. Suppose that the first k time frames of ξ have collapsed to trace $(\boldsymbol{e_0},\boldsymbol{d_0}),\dots,(\boldsymbol{e_{k-1}},\boldsymbol{d_{k-1}})$. Let $\boldsymbol{e_k}$ denote the state to which ξ transitions under input ($\boldsymbol{e_{k-1}}$,$\boldsymbol{d_{k-1}}$). The first outcome is as follows. Suppose that when eliminating an initial input pair ($\boldsymbol{e_k}$,$\boldsymbol{x}$) one of the range reduction functions excludes a bad state $\boldsymbol{e_{m+1}}$. Then one can build a trace $(\boldsymbol{e_k},\boldsymbol{d_k}),\dots,(\boldsymbol{e_m},\boldsymbol{d_m})$ leading to $\boldsymbol{e_{m+1}}$. This trace can be extended to trace $(\boldsymbol{e_0},\boldsymbol{s_0}),\dots,(\boldsymbol{e_m},\boldsymbol{s_m})$ that is a counterexample of the initial system ξ.

The second outcome is that $\boldsymbol{e_k}$ repeats a state $\boldsymbol{e_i}$, $i \leq k$. This means that the procedure of excluding initial input pairs above will be reproducing the same states between $\boldsymbol{e_i}$ and $\boldsymbol{e_k}$. So no counterexample of length at most n breaking property P exists and hence P holds for n transitions.

The third outcome is that the first n time frames are collapsed to a trace $(\boldsymbol{e_0},\boldsymbol{d_0}),\dots,(\boldsymbol{e_{n-1}},\boldsymbol{d_{n-1}})$ where all states $\boldsymbol{e_i}$ are good and different from each other. This means that property P holds for n transitions.

VI. Isolated And Public Traces

In this section, we classify the traces excluded by a clause C into two sets: isolated traces and public traces. The importance of such classification is as follows. First, as we show in Section VII, one can use CRR to efficiently find isolated counterexamples. Second, as we prove in Proposition 4 below, if C does not exclude an isolated counterexample of length at most n disproving property P, then C is a P^n-equivalent clause.

Definition 9: Let ξ be a state transition system. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let E denote an initialized trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ that is excluded by C. We will call E **isolated with respect to clause C** if no state $\boldsymbol{s_i}$,$i > 0$ of E can be reached by a trace of length i allowed by C. Otherwise, E is said to be **public with respect to clause C**.

Proposition 3: Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H_1,\dots,H_m$ be range reduction formulas computed with respect to clause C. Let E denote an initialized trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ such that

- ($\boldsymbol{s_0}$,$\boldsymbol{x_0}$) falsifies C i.e. E is excluded by C
- ($\boldsymbol{s_i}$,$\boldsymbol{x_i}$) falsifies H_i, $i = 1,\dots,m$.

Then E is isolated with respect to C.

Definition 10: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let E denote a a counterexample $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$. We will say that E is a **counterexample isolated (or public) with respect to clause C** if trace E is isolated (respectively public) with respect to C.

Proposition 4: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Assume that C does not exclude any counterexample of length at most n isolated with respect to C. Then C is a P^n-equivalent clause.

VII. Bug Hunting By CRR

In this section, we describe how bug hunting is done by CRR. In Subsection VII-A, we discuss the construction of range reduction formulas by a noise-free PQE solver. We show that such a PQE-solver can *prove the existence* of a bug without generation of an explicit counterexample. This is done by just showing that excluding initial input pairs by a clause C leads to excluding a trace of length k leading to a bad state. In Subsection VII-B, we describe building range reduction formulas by a noisy PQE-solver. We show that in this case, one has to build a counterexample *explicitly*.

A. Bug hunting with a noise-free PQE solver

Proposition 5: Let ξ be a state transition system with property P. Let $C(S,X)$ be a non-empty clause such that $\overline{C} \rightarrow I$. Let H_0 denote formula equal to C. Let formulas $H_1,\dots,H_n$ be obtained recursively as follows. Let Φ_0 denote formula equal to I. Let Φ_i, $0 < i \leq n$ denote formula $I \wedge H_0 \wedge T_0 \wedge \dots \wedge H_{i-1} \wedge T_{i-1}$. Here $T_j = T(S_j,X_j,S_{j+1})$ where S_j and X_j are state and input variables of j-th time frame. Formula H_{i+1} is obtained by taking H_i out of the scope of quantifiers in formula $\exists W[H_i \wedge T_i \wedge \Phi_i]$ where $W = S_0 \cup X_0 \cup \dots \cup S_i \cup X_i$. That is $H_{i+1} \wedge \exists W[T_i \wedge \Phi_i] \equiv \exists W[H_i \wedge T_i \wedge \Phi_i]$. Then formulas $H_1,\dots,H_n$ are range reduction formulas.

The fact that range reduction formula H_i excludes only reachable states guarantees that if $\overline{H}_i \not\rightarrow P$ then a counterexample exists. As we show below this is not true when a noisy PQE solver is used.

B. Bug hunting with a noisy PQE solver

Proposition 6: Let $H_i^*, i = 0,\dots,n$ be formulas obtained as described in Proposition 5 with only one exception. A *noisy* PQE-solver is used to obtain H_{i+1}^* by taking H_i^* out of the scope of quantifiers in $\exists W[H_i^* \wedge T_i \wedge \Phi_i^*]$. Here $\Phi_0^* = I$, $H_0^* = C$ and $\Phi_i^* = I \wedge H_0^* \wedge T_0 \wedge \dots \wedge H_{i-1}^* \wedge T_{i-1}$ for $i < 0 \leq n$. Then $H_i^* \rightarrow H_i$ holds where $H_i, i = 1,\dots,n$ are range reduction formulas.

Proposition 6 suggests that a noisy PQE-solver, in general, builds a formula H_i^* that over-approximates the set of states for which a correct range reduction formula H_i evaluates to 0. For that reason we will refer to H_i^* as an **approximate range reduction formula**. Since H_i^* is not logically equivalent to H_i, the former can exclude states that are not reachable by ξ. So if $\overline{H_i^*} \not\rightarrow P$ for some state $\boldsymbol{s_i}$, one needs to check if $\boldsymbol{s_i}$ is reachable from an initial state. This can be done as follows. First, a state $\boldsymbol{s_{i-1}}$ from which there is a transition to $\boldsymbol{s_i}$ is searched for. If such a state exists, then a state $\boldsymbol{s_{i-2}}$ from which there is a transition to $\boldsymbol{s_{i-1}}$ is searched for and so on. This process results either in finding a counterexample reaching state $\boldsymbol{s_i}$ or deriving a clause falsified by $\boldsymbol{s_i}$. The latter means that $\boldsymbol{s_i}$ is unreachable.

C. Building range reduction formulas incrementally

In the previous subsection, we showed that a range reduction formula H_{i+1} can be obtained by taking H_i out of the scope of quantifiers in $\exists W[H_i \wedge T_i \wedge \Phi_i]$. Note that formula Φ_i contains $i-1$ copies of the transition relation and so gets very large as i grows. Fortunately, in general, one only needs a small set of time frames preceding the time frame i to derive the clauses of H_{i+1}. This makes derivation of H_{i+1} *local*.

The reason for derivation of H_{i+1} to be local is as follows. Solving the PQE problem comes down to generating a set of clauses depending on free variables of $\exists W[H_i \wedge T \wedge \Phi_i]$ that makes the clauses of H_i redundant in $\exists W[H_i \wedge T \wedge \Phi_i]$. In [2], we introduced a PQE-solver called *DS-PQE* that implements this strategy. To solve the PQE problem, *DS-PQE* maintains a set of clauses to be Proved Redundant. We will refer to a clause of this set as a PR-clause. Originally, the set of PR-clauses consists of the clauses of H_i. A resolvent clause C that is a descendant of H_i also becomes a PR clause and so needs to be proved redundant. The only exception is the case when C depends only of free variables of $\exists W[H_i \wedge T \wedge \Phi_i]$ i.e. on variables of S_{i+1}. Then C is just added to H_{i+1}.

DS-PQE uses branching to first prove redundancy of PR-clauses in subspaces. Then it merges the results of different branches. Importantly, *DS-PQE* backtracks as soon as all PR-clauses are proved redundant in the current subspace. This means that *DS-PQE* needs clauses of Φ_i corresponding to j-th time frame where $j < i$ only if there is a PR-clause that contains a variable of j-th time frame. *DS-PQE* produces a new PR-clause C'' obtained from another PR-clause *only* if it cannot prove redundancy of C' in the current subspace. Generation of C'' can be avoided if the current formula has a non-PR clause that subsumes C' in the current subspace. For instance, one could prevent the appearance of clause C'' containing a variable of j-th time frame by exploiting non-PR clauses that depend on variables of S_{j+1} and implied by formula Φ_{j+2}. Such clauses could have been derived when building range reduction formula H_{j+2} by taking H_{j+1} out of the scope of quantifiers.

So, making computation of H_{i+1} local comes down to preventing the appearance of PR-clauses containing variables of time frames that are far away from the i-th time frame. This is achieved by re-using clauses derived when building range reduction formulas H_j, $j \leq i$.

VIII. Identification Of P-Equivalent Clauses

In this section, we describe two cases where one can prove that a clause C is P^n-equivalent. We assume here that computation of range reduction formulas is performed by a noisy PQE-solver. Proposition 7 describes the case where one needs to compute all formulas H_i^*, $i = 1,\dots,n$. The case where this is not necessary is addressed by Proposition 8.

Proposition 7: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H_1^*,\dots,H_n^*$ be approximate range reduction formulas computed with respect to clause C by a noisy PQE solver. Suppose that no formula $H_i^*, i = 1,\dots,n$ excludes a reachable bad state $\boldsymbol{s}$. Then clause C is P^n-equivalent.

Proposition 8: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H_1^*,\dots,H_i^*$ be approximate range reduction formulas computed with respect to clause C by a noisy PQE solver. Suppose that every bad state excluded by H_j^*, $1 \leq j < i$ is unreachable. Suppose that every state (bad or good) excluded by H_i^* is unreachable. Then clause C is P^n-equivalent for any $n > 0$.

Proposition 8 suggests that one can declare a clause C P^n-equivalent for an arbitrary n if formula H_i^* does not exclude any reachable states. Let us consider the following three cases. The first case is that H_i^* is empty i.e. $H_i^* \equiv 0$. The second case, is that H_i^* excludes only bad states i.e. $\overline{H_i^*} \rightarrow \overline{P}$. The third case occurs when $\overline{H_i^*} \not\rightarrow \overline{P}$ i.e. when H_i^* excludes good states.

From the viewpoint of performance, it makes sense to check if every state excluded by H_i^* is unreachable only in the first and second cases. In the first case, no state is excluded by H_i^* and so no extra work needs to be done to apply Proposition 8. In the second case, one needs to perform only one extra check to verify if $\overline{H_i^*} \rightarrow \overline{P}$ holds. Checking if a *bad* state excluded by H_i^* is reachable has to be done anyway to guarantee that no counterexample excluded by clause C is overlooked. On the other hand, if $\overline{H_i^*} \rightarrow \overline{P}$ does not hold i.e. if the third case above occurs, the amount of extra work one has to do can be very high. This is because H_i^* can exclude a large number of unreachable *good* states.

IX. Comparison Of Model Checking By CRR With Other Approaches

CRR is essentially a method of computing an *under-approximation* of the set of reachable states. In this section, we compare the CRR method with Bounded Model Checking (BMC) and with methods based on computing an *Over-approximation* of Reachable States. We will refer to the latter as ORS-methods. We will assume that the precise computation of a set of reachable states is just a special case of its over-approximation.

In Section II, we already made some comparison of the CRR method with other model checkers on a simple example. Here, we continue this work in the general case. Since, in this paper, we emphasize the great potential of using CRR in bug hunting we compare the CRR method with BMC and ORS-methods in the context of generation of counterexamples. In Subsection IX-A, we compare the CRR method and ORS-methods. In Subsection IX-B we relate the bug hunting of the CRR method with that of BMC. For the sake of simplicity, in this section, we assume that the CRR method employs a noise-free PQE-solver.

A. CRR and ORS-methods

The difference between the CRR method and ORS-methods as far as bug hunting is concerned is that the ORS-methods look for a bug in a breadth-first manner. In particular, they try to find the shortest possible counterexample. For example, IC3 first makes sure that a set of invariants is met that guarantees that no counter-example of length n exists before it increments n by 1. The reason for such strategy is that the number of states reachable in n transitions exponentially grows in n. This cripples the performance of model checkers that compute the set of reachable sets precisely. The best model checkers like IC3 address this problem by over-approximating the set of reachable states. However, finding bugs in a breadth-first manner may render inefficient even successful ORS-methods like IC3.

In contrast, the CRR-method looks for a bug in a *depth-first manner*. Given a clause $C(S,X)$ and a number n it computes a set of range reduction formulas $H_1,\dots,$. This computation goes on until a formula H_i, $i \leq n$ excludes a bad state or C is proved P^n-equivalent. A remarkable fact here is that the CRR method can *lock onto* a counterexample (that is isolated with respect C) by computing a drastic under-approximation of the set of reachable states. By locking onto a counterexample $E = (\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ we mean generation of sets D_i, $i = 0,\dots,m$ such that $D_0 \times \dots \times D_m$ contains the tuple $(\boldsymbol{s_0},\dots,\boldsymbol{s_m})$.

As we mentioned in Subsection V-A, the set of states excluded by formula H_i can be represented as the difference of sets A_i and B_i. Here A_i is the set of states reachable by traces excluded by clause C and B_i is the subset of A_i consisting of the states that are also reachable by traces allowed by C. Notice that to lock onto counterexample E, an ORS-method would have to compute sets $A_1,\dots,A_m$ or their over-approximation. The CRR method locks onto E by computing *only sets* $A_1 \setminus B_1,\dots,A_m \setminus B_m$ that can be drastically smaller.

B. CRR and BMC

Similarly to ORS-methods, BMC searches for a counterexample in a breadth-first manner. First BMC searches for a counterexample of length 1. If no such counterexample exists, BMC searches for a counterexample of length 2 and so on. So the main difference of the CRR method from BMC is that the former searches for a counterexample in a depth-first manner.

To check if a counterexample of length i exists, BMC tests the satisfiability of formula G_i equal to $I \wedge T_0 \cdots \wedge T_{i-1} \wedge \overline{P}$. The size of G_i is linear in the number of time frames. So the reach of BMC is typically limited to counterexamples of length 100-200. In theory, the CRR method has to deal with formula Φ_i whose size is linear in i. However, as we conjectured in Subsection VII-C, the computation of range formula H_{i+1} that involves formula Φ_i can be made local. In this case, only clauses of a small number of time frames preceding time frame i are employed. So the CRR method can potentially find very long counterexamples.

One more important advantage of the CRR method over BMC is that it can derive clauses that are hard or even impossible to derive by a regular SAT-solver [2]. Suppose, for example, that a clause C is proved P^i-equivalent. If C eliminates a counterexample of length i it is not implied by the formula G_i that BMC checks for satisfiability. (Because C eliminates an assignment satisfying G_i.) So clause C cannot be derived by a resolution based SAT-solver from G_i. Adding C to G_i only preserves the satisfiability of the latter.

Importantly, the CRR method derives a P^i-equivalent clause C differently from BMC even if C is implied by G_i. We assume here that the PQE-solver used by the CRR method to compute range reduction formulas employs the machinery of D-sequents [1]. Then, such a PQE-solver can produce clauses obtained by non-resolution derivation. An example of a clause obtained by non-resolution derivation is a blocked clause [5]. Adding clauses obtained by non-resolution derivation allows one to get proofs that are much shorter than those based on pure resolution. For example, in [4] it was shown that extending resolution with a rule allowing to add blocked clauses makes it exponentially more powerful.

X. Description of MC_CRR

In this section, we describe a model checker called MC_CRR that is based on CRR. To make this description simpler we omitted some obvious optimizations. For example, the current version of MC_CRR discards approximate range reduction formulas H^*_i, $i = 1,\dots,n$ computed with respect to a clause C after C is proved P^n-equivalent. Only clause C itself is kept and re-used when a new clause C' is checked for being P^n-equivalent. In reality, formulas H^*_i can be re-used as well. The same applies to formulas U_i, $i = 1,\dots,n$ generated when eliminating unreachable bad states falsifying H^*_i. In the current version of MC_CRR , these formulas are discarded after C is proved P^n-equivalent. In reality, they can

```
// N is a comb. circuit specifying transition relation
// I is a CNF formula specifying initial states
// P is the property to be checked
// n is length of the longest counterexample (if any)
// MC_CRR returns a counterexample
// or nil if no counterexample exists
//
MC_CRR(N, I, P, n){
1  T(S, X, S') := GenCnfForm(N);
2  Trace := ∅;
3  States := ∅;
   - - - - - - - - - - - - -
4  while (true) {
5     (s, x) := GenInp(I);
6     s' := Simulate(P, N, s, x);
7     if (P(s') = 0) return(Trace ∪ {(s, x)});
8     A := MinFalsifClause(s, x);
   - - - - - - - - - - - - -
9     E := ConstrTimeFrame(T, I, P, A, n);
10    if (E ≠ nil) return(Trace ∪ E);
11    Trace := Trace ∪ {(s, x)};
12    States := States ∪ s;
13    if (s' ∈ States) return(nil);
14    I := FormUnitClauses(s');}}
```

Fig. 1. Model checking by computing range reduction

be re-used when checking P^n-equivalence of a new clause C'.

A. Main procedure

The pseudo-code of MC_CRR is given in Figure 1. MC_CRR accepts a state transition system ξ described by a circuit N and predicates I and P. N specifies a transition relation and predicates I and P specifies initial states and the property to verify. MC_CRR also accepts parameter n informing the model checker that one needs to check if P holds for n transitions. The main parts of the code are separated by the dotted lines. MC_CRR starts with generating formula T specifying the transition relation represented by circuit N and initializing some variables (lines 1-3). As we mentioned earlier, MC_CRR reduces the set of all traces of length at most n to one trace. This is done by keeping only one input pair per time frame processed by MC_CRR. The set of these input pairs is accumulated in variable *Trace*. Variable *States* collects all the states of the trace stored in *Trace*. Variables *Trace* and *States* are initialized to an empty set (lines 2-3).

The main computation of MC_CRR takes place in a *while* loop (lines 4-14). The body of the loop consists of two parts. In the first part (lines 4-8) MC_CRR generates an input pair ($\boldsymbol{s}$,$\boldsymbol{x}$) that MC_CRR does not exclude from the current initial time frame (line 5). Then MC_CRR checks that property P holds for state $\boldsymbol{s'}$ to which system ξ transitions under input pair ($\boldsymbol{s}$,$\boldsymbol{x}$) (lines 6-7). If $\boldsymbol{s'}$ breaks P then MC_CRR returns $\mathit{Trace} \cup \{(\boldsymbol{s},\boldsymbol{x})\}$ as a counterexample. Otherwise, MC_CRR generates $A(S,X)$, the longest clause falsified by ($\boldsymbol{s}$,$\boldsymbol{x}$). For every clause C generated by procedure *ConstrTimeFrame* (line 9) to constrain input pairs of the initial time frame, $C \not\rightarrow A$ holds. This guarantees that C does not exclude the input pair ($\boldsymbol{s}$,$\boldsymbol{x}$).

```
// E denotes a counterexample
//
ConstrTimeFrame(T, I, P, A, n){
1  G := ∅;
2  while (true) {
3     (s, x) := SatAssgn(G ∧ A);
4     if ((s, x) = nil) return(nil);
5     C := GenFalsifClause(G, A, s, x); // C ↛ A
6     E := CompRrForm(T, I, P, C, n);
7     if (E ≠ nil) return(E);
8     G := G ∧ C; }}
```

Fig. 2. The *ConstrTimeFrame* procedure

MC_CRR starts the second part of the loop (lines 9-14) calling procedure *ConstrTimeFrame*. This procedure tries to exclude all the input pairs of the current time frame but ($\boldsymbol{s}$,$\boldsymbol{x}$). If *ConstrTimeFrame* fails to do this, it returns a trace E for which P fails. This trace does not include the time frames already processed by MC_CRR . For that reason, to form a counterexample for the original system ξ one needs to take the union of *Trace* and E. If *ConstrTimeFrame* succeeds, then either property P holds for n transitions or every counterexample contains the input pair ($\boldsymbol{s}$,$\boldsymbol{x}$) selected in line 5 that has not been eliminated. In this case, MC_CRR updates sets *Trace* and *States* by adding ($\boldsymbol{s}$,$\boldsymbol{x}$) and $\boldsymbol{s}$ respectively. Then MC_CRR checks if the state $\boldsymbol{s'}$ to which ξ transitions under input ($\boldsymbol{s}$,$\boldsymbol{x}$) is already in the set *States* (line 13). (Recall that *States* contains all the states of *Trace*.) If it is, then property P holds for n transitions because the part of *Trace* between two copies of state $\boldsymbol{s'}$ can be repeated. Finally, MC_CRR eliminates the current time frame by making $\boldsymbol{s'}$ the new initial state (line 14). So the second time frame of system ξ with initial states I becomes the new initial time frame of ξ with initial state $\boldsymbol{s'}$.

B. Constraining a time frame

The pseudo-code of procedure *ConstrTimeFrame* is shown in Figure 2. The objective of this procedure is to exclude all input pairs of the initial time frame but the input pair falsifying clause A. The set of generated P^n-equivalent clauses is accumulated in formula G that is initially empty. Computation is performed in a *while* loop. First, *ConstrTimeFrame* generates a new input pair ($\boldsymbol{s}$,$\boldsymbol{x}$) to exclude. This input pair is formed as an assignment satisfying $G \wedge A$ (line 3). If $G \wedge A$ is unsatisfiable, then G has excluded all the input pairs but the input pair falsifying A. In this case, *ConstrTimeFrame* returns *nil* meaning that no counterexample is found (line 4).

If a new input pair to exclude ($\boldsymbol{s}$,$\boldsymbol{x}$) is found, *ConstrTimeFrame* generates a clause C that is falsified by ($\boldsymbol{s}$,$\boldsymbol{x}$) (line 5). Clause C is constructed in such a way that it does not exclude the input pair falsifying A. Then procedure *CompRrForm* is called that computes range reduction formulas with respect to clause C (line 6). If *CompRrForm* returns a counterexample (line 7) it means that a bad reachable state was excluded by one of the range reduction formulas. Otherwise, C is added

// $T_j = T(S_{j-1}, X_{j-1}, S'_{j-1})$
//
$CompRrForm(T, I, P, C, n)\{$
1 $\mathbb{H}^* := \{H^*_0, \dots, H^*_n\}$;
2 $H^*_0 := \{C\}$;
3 $\mathbb{U} := \{U_1, \dots, U_n\}$;
4 $W := S_0 \cup X_0$;
5 $\Phi^* := I$;
6 for ($j = 0;\ j < n;\ j{+}{+}$) $\{$
7 $\quad H^*_{j+1} := SolvePQE(\exists W[H^*_j \wedge T_j \wedge \Phi^*])$;
8 $\quad$ if ($\overline{H^*_{j+1}} \not\rightarrow P$) $\{$
9 $\quad\quad (E, \mathbb{U}) := ElimBadStates(T, I, P, \mathbb{H}^*, \mathbb{U}, j{+}1)$;
10 $\quad\quad$ if ($E \neq nil$) return(E);$\}$
11 $\quad$ if ($\overline{H^*_{j+1}} \rightarrow \overline{U}_{j+1}$) return($nil$);
12 $\quad W := W \cup S_j \cup X_j$;
13 $\quad \Phi^* := \Phi^* \wedge H^*_j \wedge T_j$; $\}$
14 return(nil); $\}$

Fig. 3. The *CompRrForm* procedure

$ElimBadStates(T, I, P, \mathbb{H}^*, \mathbb{U}, j)\{$
1 while ($true$) $\{$
2 $\quad \boldsymbol{p}{=}SatAssgn(\overline{H^*_j} \wedge U_j \wedge \overline{P})$;
3 $\quad$ if ($\boldsymbol{p} = nil$) return($nil, \mathbb{U}$);
4 $\quad \boldsymbol{s_j} := ExtrState(\boldsymbol{p})$;
5 $\quad (E, C) := PropBack(T, I, P, \mathbb{H}^*, \mathbb{U}, \boldsymbol{s_j}, j)$;
6 $\quad$ if ($E \neq nil$) return($E, \mathbb{U}$);
7 $\quad U_j := U_j \wedge C$;$\}\}$

Fig. 4. The *ElimBadStates* procedure

// E denotes a counterexample
//
$PropBack(T, I, P, \mathbb{H}^*, \mathbb{U}, \boldsymbol{s_k}, k)\{$
1 $E := \emptyset$;
2 $j := k$;
3 while ($true$) $\{$
4 $\quad$ if ($j = 0$) return(E, nil);
5 $\quad (\boldsymbol{p}, Proof) := SatAssgn(U_{j-1} \wedge T \wedge Cnf(\boldsymbol{s_j}))$;
6 $\quad$ if ($\boldsymbol{p} = nil$) $\{$
7 $\quad\quad C := FormClause(Proof, Cnfs(\boldsymbol{s_j}))$;
8 $\quad\quad$ if ($j = k$) return(nil, C);
9 $\quad\quad U_j := U_j \wedge C$;
10 $\quad\quad E := E \setminus \{(\boldsymbol{s_j}, \boldsymbol{x_j})\}$;
11 $\quad\quad j := j + 1$;
12 $\quad\quad$ continue; $\}$
13 $\quad (\boldsymbol{s_{j-1}}, \boldsymbol{x_{j-1}}) = ExtrInpPair(\boldsymbol{p})$;
14 $\quad E := E \cup \{(\boldsymbol{s_{j-1}}, \boldsymbol{x_{j-1}})\}$;
15 $\quad j := j - 1$;$\}\}$

Fig. 5. The *PropBack* procedure

to G as a P^n-equivalent clause.

C. Computing range reduction formulas

Procedure *CompRrForm* computing a set $\mathbb{H}^*$=$\{H^*_0, \dots, \}$ of range reduction formulas is shown in Figure 3. We assume that MC_CRR employs a noisy PQE-solver. So H^*_i are approximate range reduction formulas. For the sake of simplicity, in this section we will refer to H^*_i as just range reduction formulas. Formula H^*_i specifies states that become unreachable in i transitions due to excluding the input pairs of the initial time frame that falsify clause C. Formula H^*_0 is just equal to clause C (line 2). The formulas H^*_i, $0 < i \leq n$ are initialized to an empty set of clauses. *CompRrForm* also forms the set $\mathbb{U}$=$\{U_1, \dots, U_n\}$ (line 3). Formula U_i is meant to eliminate the bad states falsifying formula H^*_i that are unreachable. In other words, U_i is meant to make up for the fact that MC_CRR uses a noisy PQE-algorithm. *CompRrForm* also initializes set W and formula Φ^* (lines 4-5). They are used in formulating PQE problems to be solved (line 7).

The main computation is performed in a *while* loop (lines 6-13). First, a range reduction formula H^*_{j+1} is computed by taking H^*_j out of the scope of quantifiers in formula $\exists W[H^*_j \wedge T_j \wedge \Phi^*]$ (line 7). Here W is equal to $S_0 \cup X_0 \cup \dots \cup S_j \cup X_j$ for $j \geq 0$ and Φ^* is equal to $I \wedge H^*_0 \wedge T_0 \dots \wedge H^*_{j-1} \wedge T_{j-1}$ for $j > 0$. Then *CompRrForm* checks if H^*_{j+1} excludes a bad state (line 8). If it does, then procedure *ElimBadStates* is called (line 9) to check if a bad state excluded by H^*_{j+1} is reachable from an initial state. If so, then *ElimBadStates* returns a trace E leading to such a bad state (line 10). In the process of checking if bad states excluded by H^*_{j+1} are reachable, *ElimBadStates* updates formulas U_i, $i = 1, \dots, j{+}1$ by adding new clauses eliminating unreachable bad states.

After all bad states excluded by H^*_{i+1} are eliminated by clauses of U_{j+1}, *CompRrForm* checks the condition of Proposition 8. Namely, it checks if all states excluded by H^*_{i+1} are eliminated as unreachable by U_{j+1}. If this is the case, *CompRrForm* returns *nil* reporting that C is P^n-equivalent (line 11). Otherwise, *CompRrForm* switches to a new time frame by updating set W and formula Φ^* (lines 12-13).

If none of the formulas H^*_i, $i = 1, \dots, n$ excludes a reachable bad state, then from Proposition 7 it follows that C is P^n-equivalent. So *CompRrForm* returns *nil* (line 14).

D. Searching for a counterexample

MC_CRR searches for a counterexample by calling procedure *ElimBadStates* (Figure 4) that, in turn, calls procedure *PropBack* (Figure 5). The objective of *ElimBadStates* is to show that the bad states excluded by a range reduction formula H^*_j are unreachable. This is done in a *while* loop. First, *ElimBadStates* checks if formula $\overline{H}_j \wedge U_j \wedge \overline{P}$ is satisfiable (line 2). Suppose that a satisfying assignment $\boldsymbol{p}$ is found. Then one can extract a bad state $\boldsymbol{s}$ from $\boldsymbol{p}$ that is excluded by the range reduction formula H^*_j and satisfies U_j (line 4). The latter means that $\boldsymbol{s}$ has not been proved unreachable yet. Then procedure *PropBack* returns a trace from an initial state to $\boldsymbol{s}$ (if any). If such a trace E exists then *ElimBadStates* terminates returning E. Otherwise, *PropBack* returns a clause C that is falsified by $\boldsymbol{s}$ thus proving that $\boldsymbol{s}$ is unreachable. Clause C is added to U_j and a new iteration starts.

The goal of procedure *PropBack* (Figure 5) is to find an initialized trace E leading to the bad state $\boldsymbol{s_k}$. Initially E is an empty set (line 1). Trace E is built in the reverse order. So index j specifying the current time frame is initialized to k (line 2). The main computation is done in a *while* loop (lines

3-15). If j is equal to 0, then the construction of E is over (line 4). Otherwise, *PropBack* checks if formula $U_{j-1} \wedge T \wedge Cnf(\boldsymbol{s_j}))$ is satisfiable. Here $Cnf(\boldsymbol{s_j})$ is the set of unit clauses specifying state $\boldsymbol{s_j}$. The existence of a satisfying assignment $\boldsymbol{p}$ means that one can extract an input pair $(\boldsymbol{s_{j-1}},\boldsymbol{x_{j-1}})$ from $\boldsymbol{p}$ such that

- $\boldsymbol{s_{j-1}}$ satisfies U_{j-1} and hence is not proved unreachable yet
- system ξ transitions to state $\boldsymbol{s_j}$ under input pair $(\boldsymbol{s_{j-1}},\boldsymbol{x_{j-1}})$.

Assume that $\boldsymbol{p}$ does not exist. In this case a resolution proof of unsatisfiability *Proof* is generated and *PropBack* performs actions shown in lines 7-12. First, a clause C falsified by $\boldsymbol{s_j}$ is built. The simplest way to construct C is to negate $Cnf(\boldsymbol{s_j})$. A shorter clause can be generated by excluding from C the literals that correspond to the unit clauses of $Cnf(\boldsymbol{s_j})$ that were not used in *Proof*. If $j = k$, then C is falsified by the target state $\boldsymbol{s_k}$ thus proving the latter unreachable (line 8). Otherwise, *PropBack* adds C to U_j. After that the input pair $(\boldsymbol{s_j},\boldsymbol{x_j})$ is removed from E, index j is incremented by 1 and a new iteration starts (lines 10-12).

If a satisfying assignment $\boldsymbol{p}$ exists, then an input pair $(\boldsymbol{s_{j-1}},\boldsymbol{x_{j-1}})$ is extracted from $\boldsymbol{p}$ and added to E (lines 13-14). The value of j is decremented by 1 and a new iteration begins (line 15).

XI. Experimental results

The key operation of MC_CRR is to compute a range reduction formula by running a PQE-solver. In this section, we describe experiments meant to show the viability of using a PQE-solver for computing range reduction. We will conduct a more thorough experimental study once MC_CRR is implemented. In the experiments, we used the PQE-algorithm called *DS-PQE* [2].

In addition to showing the viability of using PQE for computing range reduction, the experiments pursued three other goals. The first goal was to show that PQE can be much more efficient than QE. The second goal was to demonstrate that reducing the noise generated by a PQE-solver can significantly improve its performance. This third goal was to show that the set of states excluded by range reduction formulas is drastically smaller than the set of reachable states.

A. Using PQE-solver to compute range reduction

In this subsection, we describe experiments with computing range reduction by a PQE-solver. In these experiments, we used 758 benchmarks of HWMCC-10 competition. Let $N(X,S,S')$ be the circuit representing a transition relation and T be a CNF formula specifying N. Recall that X denotes the input combinational variables and S,S' denote the present and next state variables. (For the sake of simplicity we do not mention the internal combinational variables of N.) So $S \cup X$ and S' specify the input and output variables of N respectively.

In experiments, we computed the range reduction of N caused by excluding the input pairs $(\boldsymbol{s},\boldsymbol{x})$ falsifying a clause

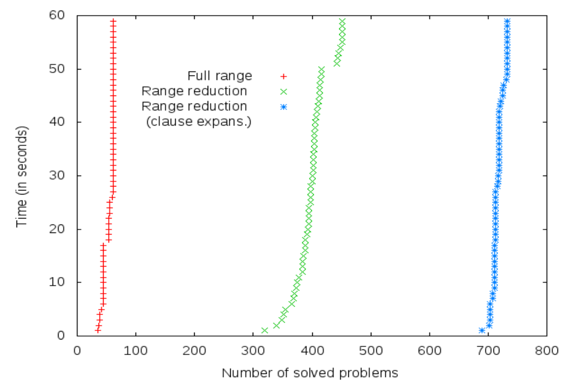

Fig. 6. Computing full range and range reduction

$C(S,X)$. We used two methods of computing range reduction. The first method was just to run *DS-PQE* on formula $\exists W[C \wedge T]$ where $W = S \cup X$. The second method first optimized clause C to reduce the amount of noise generated by *DS-PQE*. This optimization was performed by a technique called *clause expansion*. The idea of clause expansion is to replace C with a clause $C \vee lit(v)$ if the clause $C \vee \overline{lit}(v)$ is implied by T. Here $lit(v)$ is a literal of variable v. It is not hard to show that, in this case, taking out clause C from $\exists W[C \wedge T]$ is equivalent to taking out C' from $\exists W[C' \wedge T]$ where $C' = C \vee lit(v)$. The objective of replacing C with $C \vee lit(v)$ is to reduce noise generation by removing the part of C that is implied by T. Note that clause $C \vee lit(v)$ can be further expanded. So, in the second method, *DS-PQE* was applied to formula $\exists W[C^* \wedge T]$ where C^* was obtained from C by adding literals. We used a very efficient procedure of clause expansion. We omit the details of this procedure.

For every transition relation out of 758, we generated a random clause C of length $|S|$ consisting of literals of S. (Note that the total set of input variables of circuit N specifying a transition relation is $S \cup X$. So clause C excluded 2^k input assignments where $k = |X|$. In many transition relations k was greater than 100.) Then we tried to check range reduction by the two methods above. We ran many experiments generating different clauses for the same transition relation. Here are the results of a typical run consisting of 758 problems where for every transition relation one clause of $|S|$ literals was generated randomly. When using the first method, only 452 out of 758 problems were finished within the 60s time limit. The second method succeeded in 733 out of 758 problems. Most of them were finished within a second.

Let $H(S')$ and $H^*(S')$ denote a noise-free and noisy solution to the PQE problem $\exists W[C \wedge T]$ respectively. That is $H^* \wedge \exists W[T] \equiv H \wedge \exists W[T] \equiv \exists W[C \wedge T]$ and $H^* \rightarrow H$. Our PQE-solver *DS-PQE* is noisy. So it generates formula H^* rather than H. In 643 out of 733 problems solved by the second method, H^* was empty meaning that no range reduction occurred. In this case $H^* \equiv H$. In the remaining 90 solved problems, in 2 cases, $T \rightarrow H^*$ held. So here, no range reduction occurred either and *DS-PQE* just generated noise.

In 88 cases, $T \rightarrow H^*$ did not hold, indicating that the range of T reduced.

The performance of these two methods of computing range reduction in the run we described above is shown in Figure 6. This figure also provides data on computing the *full range* of transition relations for the 758 HWMCC-10 benchmarks. As we discussed in Subsection III-C, finding the full range of a combinational circuit reduces to QE. So comparing methods for computing full range and range reduction is a way to compare QE and PQE. In the experiments, we computed full range by the QE-algorithm called *DCDS* [1]. With the time limit of 60s, *DCDS* finished only for 62 transition relations.

Figure 6 shows the number of problems finished in a given amount of time. This data indicates that PQE can be dramatically more efficient that QE. One more conclusion that can be drawn from Figure 6 is that reducing noise generation, e.g. by clause expansion, can have a drastic effect on the performance of a PQE algorithm.

B. Comparing set of states describing range reduction with that of reachable states

Let N be a circuit specifying a transition relation. Let H be a noise-free formula describing the range reduction of N caused by constraining inputs with clause C. Assume that C depends only on state variables. In terms of Subsection V-A, the set of states falsifying H can be represented as $A \setminus B$. Here A consists of the states that are reachable from the states falsified by C in one transition. The set B is a subset of A. It consists of the states of A that are also reachable in one transition from states *satisfying* clause C. So $A \setminus B$ consists of states that are reachable *only* from states falsifying clause C.

In Subsection V-A, we conjectured that the set $A \setminus B$ can be dramatically smaller than sets A and B. In this subsection, we check this conjecture by comparing the size of the set $A \setminus B$ and A experimentally. Given a clause C, computing the set $A \setminus B$ comes down to finding a range reduction formula H. Formula H is obtained by taking C out of the scope of quantifiers in $\exists W[C \wedge T]$. Since *DS-PQE* produces a noisy solution H^*, we considered only the cases where H^* was empty and hence $H^* \equiv H$. Building set A comes down to finding a quantifier-free formula $G(S')$ that is logically equivalent to $\exists W[\overline{C} \wedge T]$. (So finding G reduces to the QE problem.) The set A is specified by the complete assignments to S' satisfying G. To estimate the size of A we generated a limited number of cubes containing satisfying assignments of G. The size of the largest cube was used as a lower bound on the size of set A.

To compute formula G we used our QE solver called *DCDS* mentioned above. In this experiment, we used the same 758 transition relations of the HWMCC-10 benchmark set. To make generation of formula G less trivial we generated a clause of of $0.7 \times |S|$ literals (as opposed to clauses of $|S|$ literals generated in the previous experiment).

Here are the results of a typical experiment consisting of solving 758 PQE and 758 QE problems. Every PQE problem is to take C out of the scope of quantifiers in $\exists W[C \wedge T]$. The corresponding QE problem is to eliminate quantifiers in formula $\exists W[\overline{C} \wedge T]$. With the time limit of 60s, *DS-PQE* solved 561 PQE problems while *DCDS* solved only 377 QE problems. In 490 PQE problems, an empty formulas H^* were generated i.e. set $A \setminus B$ was empty. In 347 out of these 490 cases, the corresponding QE problem was solved by *DCDS*. In 211 out of 347 cases, the size of set A was larger than 2^{30} states. In 92 out of 347 cases, the size of set A was larger than 2^{100} states.

TABLE I
Estimating the size of set A for some concrete examples

benchmark	#X-in-puts	#lat-ches	#gates	PQE (s.)	QE (s.)	size of A
brpp1neg	86	138	1,244	0.01	0.01	$> 2^{84}$
eijks1423	17	157	1,101	0.01	18	$> 2^{31}$
bc57sensorsp0	97	167	1,691	0.01	0.4	$> 2^{105}$
irstdme6	220	245	1,713	0.01	0.2	$> 2^{101}$
csmacdp0neg	146	265	5,247	0.01	13	$> 2^{169}$
139452p24	225	314	5,867	0.03	0.1	$> 2^{240}$
pj2013	1,305	1,271	35,630	0.2	0.1	$> 2^{1231}$
neclaftp1001	32	7,880	63,383	0.3	3.8	$> 2^{2252}$

Some concrete results are shown in Table I. The first column gives benchmark names. The next three columns specify the size of the circuit N specifying a transition relation, the column *#X-inputs* giving the number of combinational inputs of N. The next two columns give the time taken to solve the corresponding PQE and QE problems in seconds. The last column provides the lower bound on the size of set A. For all the examples listed in Table I the set $A \setminus B$ was empty. The results of Table I show that the size of the set $A \setminus B$ can be very small even when sets A and B are very large.

ACKNOWLEDGMENT

This research was supported in part by DARPA under AFRL Cooperative Agreement No. FA8750-10-2-0233 and by NSF grants CCF-1117184 and CCF-1319580.

XII. CONCLUSIONS

We presented a new method of model checking based on the idea of Computing Range Reduction (CRR). The CRR method repetitively applies an operation that reduces the set of possible behaviors. Given a number n and a property P, the CRR method finds a counterexample of length at most n or proves that such a counterexample does not exit. A key feature of the CRR method is that it has a natural way to do bug hunting in a depth-first manner.

The results of this paper lead to the following conclusions. First, computing an *under-approximation* of available behaviors is complementary to current methods that over-approximate such behaviors. Computing an under-approximation seems to be a reasonable idea in case of bug-hunting, because one of the main concerns here is to reduce the search space. Second, a successful bug hunting tool should be able to efficiently perform depth-first search. Third, the scalability issues of model checkers based on Quantifier Elimination (QE) are caused by the fact that QE is an inherently hard problem and should be avoided. The key operation of the CRR method is based on *partial* QE. In many cases, the partial QE problem can be solved dramatically more efficiently than its QE counterpart. Fourth, the partial QE problem cannot be efficiently solved by a typical SAT-solver based on the notion of logical inconsistency rather than unobservability. So, development of non-SAT methods of model checking is very important.

APPENDIX

The appendix contains proofs of the propositions listed in the paper. We also give proofs of lemmas used in the proofs of propositions.

PROPOSITIONS OF SECTION III: PARTIAL QUANTIFIER ELIMINATION

Proposition 1: Let $R(Z)$ be a CNF formula such that $R \equiv \exists W[G]$ where $W = X \cup Y$. (That is R is a solution to the QE problem.) Then the assignments satisfying $R(z)$ specify the range of M.

Proof: Let us show that R indeed specifies the range of M. Let $\boldsymbol{z}$ be a complete assignment to Z that is in the range of M. Then there is an assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfying G and hence $\exists W[G]$ evaluates to 1 when variables Z are assigned as in $\boldsymbol{z}$. Hence $R(\boldsymbol{z})$ has to be equal to 1. Now assume that $\boldsymbol{z}$ is not in the range of M. Then no assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfies G. So $\exists W[G]$ evaluates to 0 for assignment $\boldsymbol{z}$. Then $R(\boldsymbol{z})$ has to be equal to 0. ■

Proposition 2: Let $C(X)$ be a clause depending only on input variables of M. Let $H(Z)$ be a CNF formula such that $H \wedge \exists W[G] \equiv \exists W[C \wedge G]$ where $W = X \cup Y$. (That is H is a solution to the PQE problem.) Let H and H^* be a noise-free and noisy solution respectively. Then

1) The assignments *falsifying* H specify the range reduction in M caused by excluding inputs falsifying C. That is $H(\boldsymbol{z}) = 0$ iff
   - there is an input $\boldsymbol{x}$ for which circuit M produces output $\boldsymbol{z}$
   - all inputs for which M produces output $\boldsymbol{z}$ falsify C
2) $H^* \rightarrow H$

Proof: **First condition.** Let us prove that H indeed specifies the range reduction of M. Let $\boldsymbol{z}$ be a complete assignment to Z that is in the range of M. Assume that $\boldsymbol{z}$ remains in the range of M even if the inputs falsifying clause C are excluded. Then there is an assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfying $C \wedge G$ and hence $\exists W[C \wedge G]$ evaluates to 1 when variables of Z are assigned as in $\boldsymbol{z}$. So, $H(\boldsymbol{z})$ has to be equal to 1.

Now assume that $\boldsymbol{z}$ is in the range of M but it is not in the range of M if the inputs falsifying clause C are excluded. Then no assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfies $C \wedge G$ and hence $\exists W[C \wedge G]$ evaluates to 0 when variables of Z are assigned as in $\boldsymbol{z}$. On the other hand, since $\boldsymbol{z}$ is in the range of M, there is an assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfying G. So formula $\exists W[G]$ is equal to 1 when variables of Z are assigned as in $\boldsymbol{z}$. Since $H \wedge \exists W[G]$ is equal to 0 when variables of Z are assigned as in $\boldsymbol{z}$, then $H(\boldsymbol{z})$ has to be equal to 0.

Now assume that $\boldsymbol{z}$ is not in the range of M. Then no assignment $(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ satisfies G. So both $\exists W[G]$ and $\exists W[C \wedge G]$ evaluate to 0. This means that the value of $H(\boldsymbol{z})$ is, in general, not defined. However, since we require H to be a noise-free solution, $H(\boldsymbol{z})$ has to be equal to 1.

Second condition. As we showed above, any solution to the PQE problem is defined uniquely for a complete assignment $\boldsymbol{z}$ to Z that is in the range of M. So in this case, $H(\boldsymbol{z}) = H^*(\boldsymbol{z})$. If $\boldsymbol{z}$ is not in the range of M, by definition of a noise-free solution, $H(\boldsymbol{z}) = 1$. So $H(\boldsymbol{z}) = 0$ implies $H^*(\boldsymbol{z}) = 0$ and hence $H^* \rightarrow H$. ■

PROPOSITIONS OF SECTION VI: ISOLATED AND PUBLIC TRACES

Lemma 1: Let $H_1,\dots,H_m$ be range reduction formulas computed with respect to clause C. Let E be an initialized trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ such that
- $(\boldsymbol{s_0},\boldsymbol{x_0})$ falsifies C i.e. E is excluded by C
- $(\boldsymbol{s_i},\boldsymbol{x_i})$ falsifies H_i, $i = 1,\dots,m$.

Let E' be an initialized trace $(\boldsymbol{s'_0},\boldsymbol{x'_0}),\dots,(\boldsymbol{s'_m},\boldsymbol{x'_m})$ that is allowed by clause C. Then $H_i(\boldsymbol{s'_i}) = 1$, $i = 1,\dots,m$ and hence $\boldsymbol{s_i} \neq \boldsymbol{s'_i}, i = 1,\dots,m$.

Proof: Since $\boldsymbol{s'_i}$ is in E', it is reachable by a trace allowed by C. From Definition 7 it follows that $H_i(\boldsymbol{s'_i}) = 1$. ■

Proposition 3: Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H_1,\dots,H_m$ be range reduction formulas computed with respect to clause C. Let E denote an initialized trace $(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ such that
- $(\boldsymbol{s_0},\boldsymbol{x_0})$ falsifies C i.e. E is excluded by C
- $(\boldsymbol{s_i},\boldsymbol{x_i})$ falsifies H_i, $i = 1,\dots,m$.

Then E is isolated with respect to C.

Proof: Assume that E is not isolated. Then there is an initialized trace $E' = (\boldsymbol{s'_0},\boldsymbol{x'_0}),\dots,(\boldsymbol{s'_m},\boldsymbol{x'_m})$ such that
- E' is allowed by C
- $\boldsymbol{s_i}$=$\boldsymbol{s'_i}$, for some i such that $1 \leq i \leq m$.

The existence of such a trace contradicts Lemma 1. ■

Proposition 4: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Assume that C does not exclude any counterexample of length at most n isolated with respect to C. Then C is a P^n-equivalent clause.

Proof: Assume the contrary i.e. every counterexample of length at most n is excluded by C and so C is not P^n-equivalent. Let E=$(\boldsymbol{s_0},\boldsymbol{x_0}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$ be a counterexample of length $m \leq n$ excluded by C. By assumption, E is not isolated with respect to C. Then there is an initialized trace E' equal to $(\boldsymbol{s'_0},\boldsymbol{x'_0}),\dots,(\boldsymbol{s'_k},\boldsymbol{x'_k})$, $k \leq m$ such that
- E' is allowed by C
- $\boldsymbol{s'_k} = \boldsymbol{s_k}$.

Let E'' be a sequence of input pairs $(\boldsymbol{s'_0},\boldsymbol{x'_0}),\dots,(\boldsymbol{s'_k},\boldsymbol{x'_k})$,$(\boldsymbol{s_{k+1}},\boldsymbol{x_{k+1}}),\dots,(\boldsymbol{s_m},\boldsymbol{x_m})$. Since E'' is obtained by stitching together two traces and $\boldsymbol{s'_k} = \boldsymbol{s_k}$,

E'' is a trace. Since $\boldsymbol{s_0'}$ is an initial state, E'' is an initialized trace. Since ξ transitions to a bad state under input ($\boldsymbol{s_m}$,$\boldsymbol{x_m}$) E'' is a counterexample. Since ($\boldsymbol{s_0'}$,$\boldsymbol{x_0'}$) satisfies C, E'' is allowed by C. So C does not exclude all counterexamples of length at most n and we have a contradiction. ∎

PROPOSITIONS OF SECTION VII: BUG HUNTING BY CRR

Proposition 5: Let ξ be a state transition system with property P. Let $C(S,X)$ be a non-empty clause such that $\overline{C} \rightarrow I$. Let H_0 denote formula equal to C. Let formulas $H_1,\dots,H_n$ be obtained recursively as follows. Let Φ_0 denote formula equal to I. Let $\Phi_i,\ \ 0 < i \leq n$ denote formula $I \wedge H_0 \wedge T_0 \wedge \dots \wedge H_{i-1} \wedge T_{i-1}$. Here $T_j = T(S_j,X_j,S_{j+1})$ where S_j and X_j are state and input variables of j-th time frame. Formula H_{i+1} is obtained by taking H_i out of the scope of quantifiers in formula $\exists W[H_i \wedge T_i \wedge \Phi_i]$ where $W = S_0 \cup X_0 \cup \dots \cup S_i \cup X_i$. That is $H_{i+1} \wedge \exists W[T_i \wedge \Phi_i] \equiv \exists W[H_i \wedge T_i \wedge \Phi_i]$. Then formulas $H_1,\dots,H_n$ are range reduction formulas.

Proof: Let us prove this proposition by induction on i. This proposition is vacuously true for $i=0$. Assume that it holds for $i=0,\dots,n$. Let us show that then this proposition holds for $n{+}1$. That is one needs to show that H_{n+1} is a range reduction formula and hence $H_{n+1}(\boldsymbol{s})=0$ iff $\boldsymbol{s}$ is reachable in $n+1$ transitions only by traces excluded by C. Assume that H_{n+1} is not a range reduction formula. Then one needs to consider the two cases below.

A) $H_{n+1}(\boldsymbol{s})=0$ and $\boldsymbol{s}$ is not reachable by any initialized trace of length $n{+}1$. This means that $\boldsymbol{s}$ cannot be extended to satisfy formula $I \wedge T_0 \cdots \wedge T_n$. Hence $\boldsymbol{s}$ cannot be extended to satisfy formula $\Phi_n \wedge T_n \wedge H_n$. Then the clause of maximal length falsified by $\boldsymbol{s}$ is implied by Φ_n. This means that H_{n+1} is a "noisy" solution of the PQE problem and hence cannot be obtained by a noise-free PQE solver. So we have a contradiction.

B) The set of initialized traces of length $n{+}1$ reaching state $\boldsymbol{s}$ is not empty but at least one trace of this set is allowed by C. Let E be such a trace. The fact that E reaches $\boldsymbol{s}$ means that E satisfies formula $I \wedge T_0 \cdots \wedge T_n$. Since E is a trace allowed by C it also satisfies C. Moreover, E has to satisfy all the formulas H_i, $i=1,\dots,n$. Indeed, if E falsifies H_i then there is a initialized trace of length i that is allowed by C and that reaches a state excluded by H_i. This means that H_i is not a range reduction formula. So E satisfies H_i,$i=1,\dots,n$ and hence formula $H_n \wedge T_n \wedge \Phi_n$ is satisfied by E. This means that formula H_{n+1} is not implied by $H_n \wedge T_n \wedge \Phi_n$. Hence $H_{n+1} \wedge \exists W[T_n \wedge \Phi_n]$ is not equivalent to $\exists W[H_n \wedge T_n \wedge \Phi_n]$ and we have a contradiction. ∎

Lemma 2: Let F',F'',H',H'',G be CNF formulas such that

- $H' \wedge \exists X[G] \equiv \exists X[F' \wedge G]$
- $H'' \wedge \exists X[G] \equiv \exists X[F'' \wedge G]$
- $F' \rightarrow F''$

Let H',H'' be obtained by a noise-free PQE solver. Then $H' \rightarrow H''$ holds.

Proof: Let Y denote the set of free variables. Assume the contrary i.e. $H' \not\rightarrow H''$. Then there is a complete assignment $\boldsymbol{y}$ to Y such that $H'(\boldsymbol{y})=1$ and $H''(\boldsymbol{y})=0$. The latter means that

1) $\exists X[F'' \wedge G]$=0 in subspace $\boldsymbol{y}$ and so every assignment ($\boldsymbol{x}$,$\boldsymbol{y}$) falsifies $F'' \wedge G$
2) Since H'' is obtained by a noise-free PQE solver, $G \not\rightarrow C$ where C is the longest clause falsified by $\boldsymbol{y}$. So there is an assignment ($\boldsymbol{x}$,$\boldsymbol{y}$) satisfying G.

The fact that every assignment ($\boldsymbol{x}$,$\boldsymbol{y}$) falsifies $F'' \wedge G$ and that $F' \rightarrow F''$ entails that every assignment ($\boldsymbol{x}$,$\boldsymbol{y}$) falsifies $F' \wedge G$ as well. So $\exists X[F' \wedge G]$=0 in subspace $\boldsymbol{y}$. This means that $H' \wedge \exists X[G]=0$ in subspace $\boldsymbol{y}$ as well. The fact that there is an assignment ($\boldsymbol{x}$,$\boldsymbol{y}$) satisfying G and H' depends only on variables of Y implies that $H'(\boldsymbol{y})=0$. So we have a contradiction. ∎

Proposition 6: Let $H^*_i, i=0,\dots,n$ be formulas obtained as described in Proposition 5 with only one exception. A *noisy* PQE-solver is used to obtain H^*_{i+1} by taking H^*_i out of the scope of quantifiers in $\exists W[H^*_i \wedge T_i \wedge \Phi^*_i]$. Here $\Phi^*_0 = I$, $H^*_0 = C$ and $\Phi^*_i = I \wedge H^*_0 \wedge T_0 \wedge \dots \wedge H^*_{i-1} \wedge T_{i-1}$ for $i < 0 \leq n$. Then $H^*_i \rightarrow H_i$ holds where $H_i, i=1,\dots,n$ are range reduction formulas.

Proof: We prove this proposition by induction on i. $H^*_0 \rightarrow H_0$ holds because $H^*_0 = H_0 = I$. Now we prove that $H^*_i \rightarrow H_i$, $i \geq 0$ entails that $H^*_{i+1} \rightarrow H_{i+1}$. Denote by Q_{i+1} a noise-free formula obtained by taking H^*_i out of the scope of quantifiers in $\exists W[H^*_i \wedge T_i \wedge \Phi_i]$. From Lemma 2 it follows that $Q_{i+1} \rightarrow H_{i+1}$. On the other hand, from Proposition 2 it follows that $H^*_{i+1} \rightarrow Q_{i+1}$. Hence $H^*_{i+1} \rightarrow H_{i+1}$. ∎

PROPOSITIONS OF SECTION VIII: GENERATION OF P-EQUIVALENT CLAUSES

Proposition 7: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H^*_1,\dots,H^*_n$ be approximate range reduction formulas computed with respect to clause C by a noisy PQE solver. Suppose that no formula H^*_i, $i=1,\dots,n$ excludes a reachable bad state $\boldsymbol{s}$. Then clause C is P^n-equivalent.

Proof: From Proposition 6 it follows that $H^*_i \rightarrow H_i$ where H_i is a range reduction formula. So that fact that H^*_i does not exclude a bad reachable state implies that H_i does not exclude a bad state. This means that clause C does not exclude an isolate counterexample of length at most n. Then Proposition 4 entails that C is P^n-equivalent. ∎

Proposition 8: Let ξ be a system with property P. Let $C(S,X)$ be a clause such that $\overline{C} \rightarrow I$. Let $H^*_1,\dots,H^*_i$ be approximate range reduction formulas computed with respect to clause C by a noisy PQE solver. Suppose that every bad state excluded by H^*_j, $1 \leq j < i$ is unreachable. Suppose that every state (bad or good) excluded by H^*_i is unreachable. Then clause C is P^n-equivalent for any $n > 0$.

Proof: From Proposition 6 it follows that $H^*_j \rightarrow H_j$ where H_j is a range reduction formula. So that fact that H^*_j, $1 \leq j < i$ does not exclude a bad reachable state implies that H_j does not exclude a bad state. The fact that every state excluded by H^*_i is unreachable means that H_i is empty i.e. $H_i \equiv 1$. Formula H_{i+1} is obtained by taking H_i out of the scope of quantifiers in formula $\exists W[H_i \wedge T_i \wedge \Phi_i]$. This means that $H_{i+1} =\equiv 1$ and hence H_{i+1} does not exclude any bad states either. So no formula H_i, $i > 0$ excludes a bad state. Hence clause C is P^n equivalent for any $n > 0$. ■

References

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

[2] E.Goldberg and P.Manolios. Partial quantifier elimination. Technical Report arXiv:1407.4835 [cs.LO], 2014.

[3] E.Goldberg and P.Manolios. Software for quantifier elimination in propositional logic. In *ICMS-2014,Seoul, South Korea, August 5-9*, pages 291–294, 2014.

[4] O. Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 34:73–95, 1997.

[5] O. Kullmann. New methods for 3-sat decision and worst-case analysis. *Theor. Comput. Sci.*, 223(1-2):1–72, 1999.