Safety Model Checking with Complementary Approximations

Jianwen Li*, Shufang Zhu†, Yueling Zhang†, Geguang Pu† and Moshe Y. Vardi*
*Rice University, Houston, TX, USA
†East China Normal University, Shanghai, China

***Abstract*—Formal-verification techniques, such as model checking, are becoming popular in hardware design. SAT-based model checking techniques, such as IC3/PDR, have gained a significant success in the hardware industry. In this paper, we present a new framework for SAT-based safety model checking, named *Complementary Approximate Reachability* (CAR). CAR is based on standard reachability analysis, but instead of maintaining a single sequence of reachable-state sets, CAR maintains two sequences of over- and under- approximate reachable-state sets, checking safety and unsafety at the same time. To construct the two sequences, CAR uses standard Boolean-reasoning algorithms, based on satisfiability solving, one to find a satisfying cube of a satisfiable Boolean formula, and one to provide a minimal unsatisfiable core of an unsatisfiable Boolean formula. We applied CAR to 548 hardware model-checking instances, and compared its performance with IC3/PDR. Our results show that CAR is able to solve 42 instances that cannot be solved by IC3/PDR. When evaluated against a portfolio that includes IC3/PDR and other approaches, CAR is able to solve 21 instances that the other approaches cannot solve. We conclude that CAR should be considered as a valuable member of any algorithmic portfolio for safety model checking.**

I. Introduction

Model checking is a fundamental methodology in formal verification and has received more and more concern in the hardware design community [1], [2]. Given a system model M and a property P, model checking answers the question whether P holds for M. When P is a linear-time property, this means that we check that all behaviors of M satisfy P, otherwise a violating behavior is returned as a counterexample. In the recent hardware model checking competition (HWMCC) [3], many benchmarks are collected from the hardware industry. Those benchmarks are modeled by the *aiger* format [4], in which the hardware circuit and properties (normally the outputs of the circuit) to be verified are both included. For safety checking, it answers the question whether the property (output) can be violated by feeding the circuit an arbitrary (finite) sequence of inputs. In this paper, we focus on the topic of safety model checking.

Popular hardware model checking techniques include Bounded Model Checking (BMC) [5], Interpolation Model Checking (IMC) [6] and IC3/PDR [7], [8]. BMC reduces the search to a sequence of SAT calls, each of which corresponds to the checking in a certain step. The satisfiability of one of such SAT calls proves the violation of the model to the given property. IMC combines the use of *Craig Interpolation* as an abstraction technique with the use of BMC as a search technique. IC3/PDR starts with an over-approximation, gradually then refined to be more and more precise [7], [8]. All of the three approaches have proven to be highly scalable, and are today parts of the algorithmic portfolio of modern symbolic model checkers, e.g. ABC [9].

We present here a new SAT-based model checking framework, named *Complementary Approximate Reachability* (CAR), which is motivated both by classical symbolic reachability analysis and by IC3/PDR as an abstraction-refinement technique. While standard reachability analysis maintains a single sequence of reachable-state sets, CAR maintains two sequences of over- and under-approximate reachable-state sets, checking safety and unsafety at the same time. While IC3/PDR also checks safety and unsafety at the same time, CAR does this more directly by keeping an over-approximate sequence for safety checking, and an under-approximate sequence for unsafety checking. To compute these sequences, CAR utilizes off-the-shelf Boolean-reasoning techniques for computing *Minimal Unsat Core* (MUC) [10], in order to refine the over-approximate sequence, and *Minimal Satisfying Cube* (i.e., partial assignment) [11], in order to extend the under-approximate sequences. In contrast, IC3/PDR uses a specialized technique, called *generalization*, to compute *Minimal Inductive Clauses* (MIC) [7]. Thus, IC3/PDR computes *relatively-inductive* clauses to refine the over- approximate state sequence, while CAR does not. Because of this difference, CAR and IC3/PDR are complementary, with CAR faster on some problem instances where refining by non-relatively-inductive clauses is better, and IC3/PDR faster on others where refining by relatively-inductive clauses is better.

To evaluate the performance of CAR, we benchmarked it on 548 problem instances from the 2015 Hardware Model-Checking Competition, and compared the results with IC3/PDR. The results show that while the performance of CAR does not dominate the performance of IC3/PDR, CAR complements IC3/PDR and is able to solve 42 instances that IC3/PDR cannot solve. When evaluated against a portfolio that includes IC3/PDR, BMC, and IMC, CAR is able to solve 21 instances that the other approaches cannot solve. It is well known that there is no "best" algorithm in model checking; different algorithms perform differently on different problem instances [12], and a state-of-the-art tool must implement a portfolio of different algorithms, cf. [9]. Our empirical results also support

The full version is available at https://arxiv.org/abs/1611.04946.
Geguang Pu is the corresponding author.

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

the conclusion that CAR is an important contribution to the algorithmic portfolio of symbolic model checking.

II. Preliminaries

A. Boolean Transition System, Safety Verification and Reachability Analysis

A *Boolean transition system* Sys is a tuple (V, I, T), where V is a set of Boolean variables, and every state s of the system is in 2^V, the set of truth assignments to V. I is a Boolean formula representing the set of *initial* states. Let V' be the set of primed variables (a new copy) corresponding to the variables of V, then T is a Boolean formula over $V \cup V'$, denoting the transition relation of the system. Formally, for two states $s_1, s_2 \in 2^V$, s_2 is a successor state of s_1, denoted as $(s_1, s_2) \in T$, iff $s_1 \cup s_2' \models T$, where s_2' is a primed version of s_2.

A path (of length k) in Sys is a finite state sequence $s_1, s_2, \dots, s_k$, where each $(s_i, s_{i+1})(1 \leq i \leq k-1)$ is in T. We use the notation $s_1 \to s_2 \to \dots \to s_k$ to denote a path from s_1 to s_k. We say that a state t is *reachable from* a state s, or that s *reaches* t, if there is a path from s to t. Moreover, we say t is reachable from s *in* i *steps* (resp., *within* i *steps*) if there is a path from s to t of length i (resp., of length at most i).

Let $X \subseteq 2^V$ be a set of states in Sys. We define $R(X) = \{s' | (s, s') \in T \textit{ where } s \in X\}$, i.e., $R(X)$ is the set of successors of states in X. Conversely, we define $R^{-1}(X) = \{s | (s, s') \in T \textit{ where } s' \in X\}$, i.e., $R^{-1}(X)$ is the set of predecessors of states in X. Recursively, we define $R^0(X) = X$ and $R^i(X) = R(R^{i-1}(X))$ for $i > 0$. The notations of $R^{-i}(X)$ is defined analogously.

Given a Boolean transition system $Sys = (V, I, T)$ and a safety property P, which is a Boolean formula over V, the system is called *safe* if P holds in all reachable states of Sys, and otherwise it is called *unsafe*. The safety checking asks whether Sys is safe. For unsafe systems, we want to find a path from an initial state to some state s that violates P, i.e., $s \in \neg P$. We call such state reachable to $\neg P$ a *bad* state, and the path from I to $\neg P$ a *counterexample*.

In *symbolic model checking* (SMC), safety checking is performed via symbolic reachability analysis. From the set I of initial states, we compute the set of reachable states by computing $R^i(I)$ for increasing values of i. We can compute the set of states that can reach states in $\neg P$, by computing $(R^{-1})^i(\neg P)$ for increasing values of i. The first approach is called *forward* search, while the second one is called *backward* search. The formal definition of these two approaches are shown in the table below.

	Forward	Backward
Basic:	$F_0 = I$	$B_0 = \neg P$
Induction:	$F_{i+1} = R(F_i)$	$B_{i+1} = R^{-1}(B_i)$
Terminate:	$F_{i+1} \subseteq \bigcup_{0 \leq j \leq i} F_j$	$B_{i+1} \subseteq \bigcup_{0 \leq j \leq i} B_j$
Check:	$F_i \cap \neg P \neq \emptyset$	$B_i \cap I \neq \emptyset$

For forward search, the state set F_i is the set of states that are reachable from I in i steps. This set is computed by iteratively applying R. To find a counterexample, forward search checks at every step whether one of the bad states has been reached, i.e., whether $F_i \cap \neg P \neq \emptyset$. If a counterexample is not found, the search will terminate when $F_{i+1} \subseteq \bigcup_{0 \leq j \leq i} F_j$. For backward search, the set B_i is the set of states that can reach $\neg P$ in i steps. The workflow of backward search is analogous to that of forward search. Note that forward checking of $Sys = (V, I, T)$ with respect to P is equivalent to backward checking of $Sys^{-1} = (V, \neg P, T^{-1})$ with respect to $\neg I$, where T^{-1} is simply T, with primed and unprimed variables exchanged.

B. Notations

Each variable $a \in V$ is called an *atom*. A *literal* l is an atom a or a negated atom $\neg a$. A conjunction of a set of literals, i.e., $l_1 \wedge l_2 \wedge \dots \wedge l_k$, for $k \geq 1$, is called a *cube*. Dually, a disjunction of a set of literals, i.e., $l_1 \vee l_2 \vee \dots \vee l_k$, for $k \geq 1$, is called a *clause*. Obviously, the negation of a cube is a clause, and vice versa. Let C be a set of cubes (resp., clauses), we define the Boolean formula $f(C) = \bigvee_{c \in C} c$ (resp., $f(C) = \bigwedge_{c \in C} c$). For simplicity, we use C to represent $f(C)$ when it appears in a Boolean formula; for example, the formulas $\phi \wedge C$ and $\phi \vee C$, abbreviate $\phi \wedge f(C)$ and $\phi \vee f(C)$.

A cube (/clause) c can be treated as a set of literals, a Boolean formula, or a set of states, depending on the context it is used. If c appears in a Boolean formula, for example, $c \Rightarrow \phi$, it is treated as a Boolean formula. If we say a set c_1 is a subset of c_2, then we treat c_1 and c_2 as literal sets. If we say a state st is in c, then we treat c as a set of states.

We use $s(x)/s'(x')$ to denote the current/primed version of the state s. Similarly, we use $\phi(x)/\phi'(x')$ to denote the current/primed version of a Boolean formula ϕ. For the transition formula T, we use the notation $T(x, x')$ to highlight that it contains both current and primed variables. Consider a Boolean formula ϕ whose alphabet is $V \cup V'$ and is in the conjunctive normal form (CNF). If ϕ is satisfiable, there is a full assignment $A \in 2^{V \cup V'}$ such that $A \models \phi$. Moreover, there is a *partial assignment* $A^p \subseteq A$ such that for every full assignment $A' \supseteq A^p$ it holds that $A' \models \phi$. In the following, we use the notation $pa(\phi)$ to represent a partial assignment of ϕ, and use $pa(\phi)|_x$ to represent the subset of $pa(\phi)$ achieved by projecting variables only to V. On the other hand, if ϕ is unsatisfiable, there is a *Minimal Unsat Core* (MUC) $C \subseteq \phi$ (here ϕ is treated as a set of clauses) such that C is unsatisfiable and every $C' \subset C$ is satisfiable. In the following, we use the notation $muc(\phi)$ to represent such a MUC of ϕ, and use $muc(\phi)|_{c'}$ to represent the subset of $muc(\phi)$ achieved by projecting clauses only to c'. Since c' is a cube, $muc(\phi)|_{c'}$ is also a cube.

III. The Framework of CAR

We present here a variant of standard reachability checking, in which the set of maintained states is allowed to be approximate. The new approach is named *Complementary Approximate Reachability*, abbreviated as CAR. As in standard reachability analysis, CAR also enables both forward and backward search. In the following, we introduce the forward approach in detail; the backward approach can be derived symmetrically.

A. Approximate State Sequences

In standard forward search, described in Section II, each F_i is a set of states that are reachable from I in i steps. To compute elements in F_{i+1}, previous SAT-based symbolic-model-checking approaches consider the formula $\phi = F_i(x) \wedge T(x, x')$, and use partial-assignment techniques to obtain all states in F_{i+1} from ϕ (by projecting to the prime part of the assignments). Since the set of reachable states is computed accurately, maintaining a sequence of sets of reachable states from I enables to check both safety and unsafety. However in Forward CAR, two sequences of sets of reachable states are necessary to maintain: 1). $(F_0, F_1, \ldots)$ is a sequence of over-approximate state sets, which are supersets of reachable states from I. 2) $(B_0, B_1, \ldots)$ is a sequence of under-approximate state sets, which are subsets of reachable states to $\neg P$. Under the approximation, the first sequence is only sufficient to check safety, and the second one is then required to check unsafety. The two state sequences are formally defined as follows.

Definition 1: For a Boolean system Sys and the safety property P, the over-approximate state sequences $(F_0, F_1, \ldots, F_i)$ $(i \geq 0)$, which is abbreviated as F-sequence, and the under-approximate state sequence $(B_0, B_1, \ldots, B_k)(k \geq 0)$, which is abbreviated as B-sequence, are finite sequences of state sets such that:

Basic	$F_0 = I$	$B_0 = \neg P$
Constraint	$F_j \subseteq P (0 \leq j)$	–
Inductive	$F_{j+1} \supseteq R(F_j)(j \geq 0)$	$B_{j+1} \subseteq R^{-1}(B_j)(j \geq 0)$

For each $F_i (i \geq 0)$, we call it a frame. We also define the notation $S(F) = \bigcup_{0 \leq j \leq i} F_j$ is the set of states in the F-sequence, and $S(B) = \bigcup_{0 \leq j \leq k} B_j$ is the set of states in the B-sequence.

Note that the F- and B-sequence are not required to have the same length. Intuitively, each F_{i+1} is an over-approximate set of states that are reachable from F_i in one step, and B_{i+1} is an under-approximate set of states that are reachable to B_i in one step. As we mentioned in Section II, we overload notation and consider F_i to represent (1) a set of states, (2) a set of clauses and (3) a Boolean formula in CNF. Analogously, we consider B_i to be (1) a set of states, (2) a set of cubes and (3) a Boolean formula in DNF.

The following theorem shows that, the safety checking is preserved even if $F_i (i \geq 0)$ becomes over-approximate.

Theorem 1 (Safety Checking): A system Sys is safe for P iff there is $i \geq 0$ and an F-sequence $(F_0, F_1, \ldots, F_i, F_{i+1})$ such that $F_{i+1} \subseteq \bigcup_{0 \leq j \leq i} F_j$.

Theorem 1 is insufficient for unsafety checking, as $F_{i+1} \subseteq \bigcup_{0 \leq j \leq i} F_j$ has to prove false for every $i \geq 0$. On the other hand, the unsafety checking condition $\exists i \cdot F_i \cap \neg P \neq \emptyset$ in the standard forward reachability is not correct when F_i becomes over-approximate. Our solution is to benefit from the information stored in the B-sequence.

Theorem 2 (Unsafety Checking): For a system Sys and the safety property P, Sys is unsafe for P iff there is $i \geq 0$ and a B-sequence $(B_0, B_1, \ldots, B_i)$ such that $I \cap B_i \neq \emptyset$.

Besides, since CAR maintains two different sequences, exploring the relationship between them can help to establish the framework. The following property shows that, the states stored in F- and B- sequences are unreachable when the system Sys is safe for the property P.

Property 1: For a system Sys and the safety property P, Sys is safe for P iff there is an F-sequence such that $S(F) \cap R^{-1}(S(B)) = \emptyset$ for every B-sequence.

Property 1 suggests a direction that how we can refine the F-sequence and update the B-sequence. That is to try to make the states in these two sequences unreachable. More details are shown in the next section.

We have established the Forward CAR framework, and presented the theoretical guarantee for both safety and unsafety checking. Note that symmetrically, Backward CAR performs the same framework on $Sys^{-1} = (V, \neg P, T^{-1})$ with respect to $\neg I$, where T^{-1} is simply T with primed and unprimed variables exchanged.

B. The Framework

Unlike the standard forward reachability, which computes all states in F_{i+1} from the single formula $F_i(x) \wedge T(x, x')$, Forward CAR computes elements of F_{i+1} from different SAT calls with different inputs. Each SAT call gets as input a formula of the form $F_i(x) \wedge T(x, x') \wedge c'(x')$, where the cube c is in some B_j and c' is its primed version. If the formula is satisfiable, we are able to find a new state which is in B_{j+1}; otherwise we prove that $c \cap R(F_i) = \emptyset$, which indicates F_{i+1} can be refined by adding the clause $\neg c$. The following lemma shows the main idea of computing new reachable states to $\neg P$ and new clauses to refine F_i.

Lemma 1: Let $(F_0, F_1, \ldots)$ be an F-sequence, $(B_0, B_1, \ldots)$ be a B-sequence, cube $c_1 \in B_j (j \geq 0)$ and the formula ϕ be $F_i(x) \wedge T(x, x') \wedge {c_1}'(x')(0 \leq i)$:

1) If ϕ is satisfiable, there is a cube c_2 such that every state $t \in c_2$ is a predecessor of some state s in c_1 and $t \in F_i$. By updating $B_{j+1} = B_{j+1} \cup \{c_2\}$, the sequence is still a B-sequence.
2) If ϕ is unsatisfiable, $c_1 \cap R(F_i) = \emptyset$. Moreover, there is a cube c_2 such that $c_1 \Rightarrow c_2$ and $c_2 \cap R(F_i) = \emptyset$. By updating $F_{i+1} = F_{i+1} \cup \{\neg c_2\}$, the sequence is still an F-sequence.

In the lemma above, Item 1 suggests to add a set of states rather than a single one to the B-sequence, and similarly Item 2 suggests to refine the F-sequence by blocking a set of states rather than a single one. In both situations, it will speed up the computation. These two kinds of heuristics can be achieved by partial-assignment and MUC techniques. That is, we can set $c_2 = pa(\phi)|_x$ in Item 1, and $c_2 = muc(\phi)|_{{c_1}'}$ in Item 2. Now, we provide a general framework of CAR, which is shown in Table I.

The motivation of the computation are simply twofold: 1) Enlarge the lengths of the F- and B-sequences step by step (controlled by i in the framework); 2) For each i, update both sequences until either the unsafety is detected (Step 3(b)ii) or $S(F) \cap R^{-1}(B(F)) = \emptyset$. From Property 1,

TABLE I: The Framework of Forward CAR

1) Initially, set $B_0 = \neg P, F_0 = I$;
2) If $F_0 \cap B_0 \neq \emptyset$ or $R(F_0) \cap B_0 \neq \emptyset$, return unsafe with counterexample;
3) For $i \geq 1$,
   a) Set $F_i := P$;
   b) while $S(F) \cap R^{-1}(S(B)) \neq \emptyset$
      i) Let j be the minimal index such that $F_j \cap R^{-1}(B_k) \neq \emptyset$ for some $k \geq 0$;
      ii) If $j = 0$, return unsafe with counterexample;
      iii) Let cube $c_1 = pa(F_j(x) \wedge T(x, x') \wedge {B_k}'(x'))|_x$ (From 3(b)i c_1 must exist);
      iv) Set $B_{k+1} := B_{k+1} \cup \{c_1\}$ if B_{k+1} exists, otherwise set $B_{k+1} := \{c_1\}$;
      v) Let $\phi = F_{j-1}(x) \wedge T(x, x') \wedge {c_1}'(x')$;
      vi) If ϕ is satisfiable, let $c_2 = pa(\phi)|_x$ then assert $c_2 \not\subseteq R^{-1}(S(B))$ and set $B_{k+2} := B_{k+2} \cup \{c_2\}$ if B_{k+2} exists, otherwise set $B_{k+2} := \{c_2\}$;
      vii) If ϕ is unsatisfiable, let $c_2 = muc(\phi)|_{{c_1}'}$ then assert $\neg c_2 \not\supseteq F_j$ and set $F_j := F_j \cup \{\neg c_2\}$.
   c) If $\exists 0 \leq j \leq i \cdot F_j \subseteq \bigcup_{0 \leq m \leq j-1} F_m$, return safe;
   d) Set $i = i + 1$;

$S(F) \cap R^{-1}(S(B))$ is a necessary condition to prove safety (in Step 3c). In Step 3(b)i, we choose the minimal index because CAR aims to find a counterexample, if exists, as soon as possible. The F- and B-sequence are not extended synchronously: In each i, the F-sequence is extended only once (in Step 3a), while the B-sequence is extended more than once (in Step 3(b)iv and 3(b)vi). In Step 3c, the constraint $F_j \subseteq \bigcup_{0 \leq m \leq j-1} F_m$ can be checked by SAT solvers with the input formula ($F_j \wedge \bigwedge_{0 \leq m \leq j-1} \neg F_m$). The constraint holds iff the formula is unsatisfiable. $S(F)$ and $S(B)$ are updated by default when the F- and B-sequence are updated. The correctness and termination of the framework are guaranteed by the following theorem.

Theorem 3: Given a system Sys and a safety property P, the framework terminates with a correct result.

C. Related Work

There are two main differences between CAR and IC3/PDR. First, IC3/PDR requires the F-sequence to be monotone, while CAR does not. Because CAR keeps the F-sequence non-monotone, it does not require the *push* and *propagate* processes, which are necessary in IC3/PDR. A drawback for CAR is that additional SAT calls are needed to check safety, i.e. to find $i > 0$ such that $F_{i+1} \subseteq \bigcup_{0 \leq j \leq i} F_j$ holds. In IC3/PDR, since the F-sequence is monotone, it is easy to find such i that $F_i = F_{i+1}$ syntactically.

Another main difference between CAR and IC3/PDR is the way they refine the F-sequence. CAR utilizes the off-the-shelf MUC techniques, while IC3/PDR puts more efforts to compute MIC such that the refined F-sequence is still monotone. Moreover, MIC are relatively inductive, while clauses from MUC cannot guarantee. As a result, CAR and IC3/PDR refines the F-sequence by different kinds of clauses, and thus perform differently. Although computing relatively-inductive clauses is proved to be efficient in IC3/PDR, we show in the experiments that, CAR complements IC3/PDR on the instances that computing relatively inductive clauses is not conducive for efficient checking.

It is trivial to apply the framework of CAR in both forward and backward directions by simply reversing the direction of the model. Indeed, our implementation of CAR runs the forward and backward modes in parallel. Although in theory it is also possible to run IC3/PDR in backward mode, there is a technical issue that must be addressed. In most IC3/PDR implementations, the initial states I is considered as a single cube. This helps to save a lot of SAT calls in the process of *generalization*, in which the computed clause c must satisfy $I \wedge c$ is unsatisfiable. (When I is a cube it is reduced to checking the containment of $\neg c \subseteq I$.) But usually the set of unsafe states cannot be expressed by a single cube, which makes it more complex to run IC3/PDR in a backward mode. Indeed, the evaluation of IC3/PDR in the backward mode is still an open topic.

CAR also maintains an under-approximate state sequence (B-sequence) to check unsafety, while IC3/PDR checks unsafety “on-the-fly”. Other papers also introduced multiple state sequences. The approach of “Dual Approximated Reachability” maintains two over-approximate state sequences to check safety in both forward and backward directions [13]. In contrast, we maintain two complementary (over- and under-) approximate state sequences to check safety and unsafety at the same time. In [14], states reachable from initial states are maintained to help to handle proof-obligation generation. In contrast, the B-sequence keeps states that reach bad states. In PD-KIND [15], the idea of keeping both over- and under-approximate (F- and B-) state sequences is also introduced, and the B-sequence is used to refine the F-sequence as well. However, CAR utilizes the F-sequence for safety checking and B-squence for unsafey checking, while PD-KIND utilizes the F-sequence for unsafety checking and another “induction frame” has to be introduced for the safety checking. Moreover, CAR and PD-KIND use very different underlying techniques: CAR uses MUC and partial assignment, while PD-KIND uses interpolation and generalization.

IV. Experiments

Experimental Setup In this section, we report the results of the empirical evaluation. Our (C++) model checker *CARchecker*[1] runs CAR in both Forward and Backward modes, using Minisat [16] and Muser2 [10] as the SAT and MUC engines. The tool implements the algorithm from [11] to extract partial assignments. The performance of *CARchecker*

[1] https://github.com/lijwen2748/CARchecker

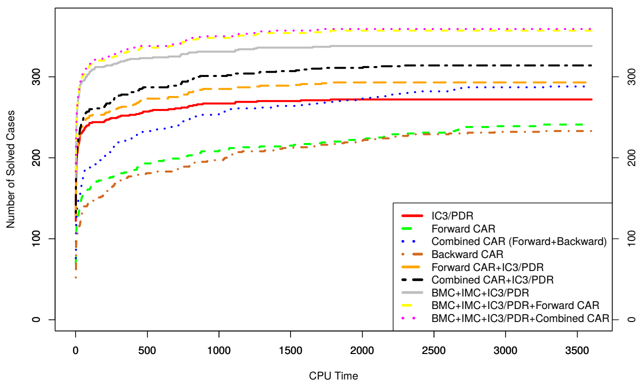

Fig. 1: Overall performance among different approaches.

is tested by evaluating it on the 548 safety benchmarks from the 2015 Hardware-Model-Checking Competition [3].

We first compared the performance of CAR with that of IC3/PDR, as implemented in the state-of-the-art model checker ABC [9] (the "pdr" command in ABC with default parameters). It should be noted that, there are many variants of IC3/PDR implementations currently, in which many heuristics are applied to the original one [17]. We choose ABC as the reference implementation for comparison because it is a standard model checker integrating several kinds of SAT-based model checking techniques. Moreover, a portfolio of modern model checkers consists of IC3/PDR, BMC (Bounded Model Checking), and IMC (Interpolation Model Checking), so we also run the experiments of BMC and IMC in ABC to explore the contribution of CAR compared to an existing portfolio (we used the "bmc2" and "int" commands in ABC with default parameters).

We run the experiments on a compute cluster that consists of 2304 processor cores in 192 nodes (12 processor cores per node), running at 2.83 GHz with 48GB of RAM per node. The operating system on the cluster is RedHat 6.0. When we run the experiments, each tool is run on a dedicated node, which guarantees that no CPU or memory conflict with other jobs will occur. Our tool *CARchecker* can run CAR in Forward mode, Backward mode or combined mode, which returns the best result from either of the approaches. In our experiments, memory limit was set to 8 GB and time limit (CPU time) to 1 hour. Instances that cannot be solved within this time limit are considered unsolved, and the corresponding time cost is set to be 1 hour. We compare the model-checking results from CARchecker with those from ABC on all benchmarks, and no discrepancy is found.

Experimental Results We show first overall performance comparison among different approaches in Fig. 1. Neither Forward CAR nor Backward CAR by itself is currently competitive with IC3/PDR. The reasons are as follows. First, the implementation of CARchecker does not utilize the power of incremental SAT computing, since the clauses to be added to the SAT solver are from the output of MUC solvers; We do not know of a way to combine them incrementally. In contrast, incremental SAT calling is an important feature in IC3/PDR. Secondly, ABC is a mature tool, incorporating many heuristics, while *CARchecker* has only been in development for a few months so it is not be surprising that ABC performs better. We believe that the performance of *CARchecker* can be improved in the future.

Nevertheless, CAR is able to compete with IC3/PDR when combining the Forward and Backward modes. In Fig. 1, the plotted line for "Combined CAR" is obtained from the best results which selected from either Forward or Backward CAR: Combined CAR solves a total number of 288 instances, while IC3/PDR solves a total number of 271 instances. Moreover, 42 instances are solved only by Combined CAR. We view the advantage of running CAR in both directions as one of the contributions of this paper; it remains to be seen whether this would also be an advantage for IC3/PDR.

Furthermore, if we consider important parameters that influence the performance, e.g., number of clauses and number of frames, we get more positive results. Note that comparing the number of SAT calls between Forward CAR and PDR is not too informative, since Forward CAR also contains MUC calls. So fewer SAT calls in Forward CAR does not mean lower cost. Fig. 2 and Fig. 3 shows respectively the scatter plots between Forward CAR and IC3/PDR on number of clauses and number of frames. From the figures, Forward CAR does not generate more clauses or more frames than IC3/PDR. In detail, 172 (175) instances are solved with fewer clauses (frames) by Forward CAR than IC3/PDR, comparing with that 121 (118) instances are solved with fewer clauses (frames) by IC3/PDR than Forward CAR. Generally speaking, the number of clauses and frames are positively correlated to the overall performance, which indicates that Forward CAR should be competitive with IC3/PDR, once CARchecker is as optimized as ABC.

Finally, we explore the contribution of CAR to the current SAT-based model-checking portfolio, which includes BMC, IMC and IC3/PDR. Fig. 1 shows the plots on the combinations IC3/PDR+BMC+IMC, IC3/PDR+BMC+IMC+Forward CAR, and IC3/PDR+BMC+IMC+Combined CAR. Forward CAR adds 19 solved instances (all safe models) to the combination of IC3/PDR+BMC+IMC, and Backward CAR solves another two (1 safe and 1 unsafe model). Although BMC solves the most unsafe cases (116), there are three unsafe instances solved only by IC3/PDR and one unsafe instance solved only by Backward CAR. For safe models, the number of solved instances only by IC3/PDR, IMC, Forward CAR and Backward CAR are 13, 12, 19, 1, respectively.

In summary, we conclude from our experimental results that 1) Forward CAR complements IC3/PDR on checking safe models; 2) Running CAR in both directions improves the performance, and 3) CAR contributes to the current portfolio of model checking strategies. We expect these conclusions to be strengthened as the development of *CARchecker* matures.

V. Concluding Remarks

CAR is inspired by IC3/PDR, but it differs from it in some crucial aspects. A main difference between CAR and IC3/PDR

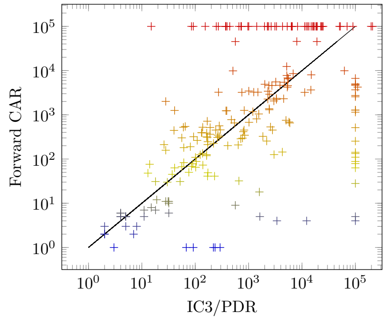

Fig. 2: Comparison on clauses.

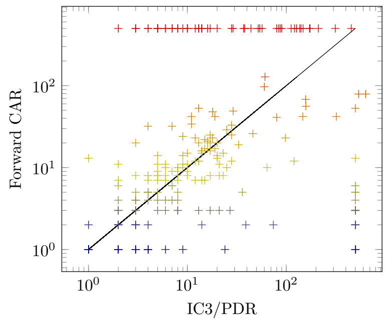

Fig. 3: Comparison on frames.

is that CAR does not require the F-sequence to be monotone. Also CAR uses a different strategy (MUC) to refine the F-sequence than IC3/PDR. Furthermore, CAR combines its over-approximate and under-approximate searches in both forward and backward modes. Due to these differences, CAR and IC3/PDR have different performance profiles. Our experiments show that IC3/PDR and CAR complement each other. The fact that our new tool, after a few months of development, outperforms mature tools that have been under development for many years over a non-negligible fraction of the benchmark suite, speaks to the merit of the new approach.

Furthermore, the area of SAT-based model checking is still a very active research topic. Many improvements to IC3/PDR have been proposed since the first published paper [7]. For example, a recent development is the combination of IC3/PDR with IMC in the Avy tool [18]. We believe that beyond the CAR tool, the CAR framework is an important contribution to this research area and will stimulate further research. For example, it may be easier to combine IMC with CAR than to combine IMC with IC3/PDR, as currently Avy has to pay extra effort to convert the generated interpolation invariants to be monotone, meeting the requirement to the state sequence maintained by IC3/PDR, while CAR does away with this monotonicity requirement.

To conclude, we presented here *Complementary Approximate Reachability* (CAR), a new framework for SAT-based safety model checking. CAR checks at the same time for both safety and unsafety in a more general way than IC3/PDR, and uses a different technique to refine the over-approximate state sequence. Experiments show that CAR complements IC3/PDR and contributes to the current portfolio, which consists of IC3/PDR, BMC and IMC. We argue therefore CAR is a promising approach for safety model checking.

VI. Acknowledgement

The authors thank anonymous reviewers for the helpful comments. Jianwen Li is partially supported by China HGJ Project (No. 2017ZX0 1038102-002), and NSFC Project No. 61532019. Geguang Pu is supported by NSFC Project No. 61572197. Moshe Y. Vardi is supported in part by NSF grants CCF-1319459 and IIS-1527668, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering".

References

[1] A. Bernardini, W. Ecker, and U. Schlichtmann, "Where formal verification can help in functional safety analysis," in *Proceedings of the 35th International Conference on Computer-Aided Design*. New York, NY, USA: ACM, 2016, pp. 85:1–85:8.

[2] A. Golnari, Y. Vizel, and S. Malik, "Error-tolerant processors: Formal specification and verification," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 286–293.

[3] A. Biere and K. Claessen, "Hardware model checking competition," .http://fmv.jku.at/hwmcc15/.

[4] A. Biere, "Aiger format," http://fmv.jku.at/aiger/FORMAT.

[5] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. 36st Design Automation Conf.* IEEE Computer Society, 1999, pp. 317–320.

[6] K. McMillan, "Interpolation and SAT-based model checking," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, J. Hunt, WarrenA. and F. Somenzi, Eds. Springer Berlin Heidelberg, 2003, vol. 2725, pp. 1–13.

[7] A. Bradley, "SAT-based model checking without unrolling," in *Verification, Model Checking, and Abstract Interpretation*, ser. Lecture Notes in Computer Science, R. Jhala and D. Schmidt, Eds. Springer Berlin Heidelberg, 2011, vol. 6538, pp. 70–87.

[8] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, 2011, pp. 125–134.

[9] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification, CAV*. Springer Berlin Heidelberg, 2010, pp. 24–40.

[10] J. Marques-Silva and I. Lynce, "On improving MUS extraction algorithms," in *Theory and Applications of Satisfiability Testing - SAT 2011*, ser. Lecture Notes in Computer Science, K. Sakallah and L. Simon, Eds. Springer Berlin Heidelberg, 2011, vol. 6695, pp. 159–173.

[11] Y. Yu, P. Subramanyan, N. Tsiskaridze, and S. Malik, "All-SAT using minimal blocking clauses," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 2014, pp. 86–91.

[12] N. Amla, X. Du, A. Kuehlmann, R. Kurshan, and K. McMillan, "An analysis of SAT-based model checking techniques in an industrial environment," in *Proc. 13th IFIG Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, ser. Lecture Notes in Computer Science, vol. 3725. Springer, 2005, pp. 254–268.

[13] Y. Vizel, O. Grumberg, and S. Shoham, "Intertwined forward-backward reachability analysis using interpolants," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2013, pp. 308–323.

[14] A. Gurfinkel and A. Ivrii, "Pushing to the top," in *Formal Methods in Computer-Aided Design.*, 2015, pp. 65–72.

[15] D. Jovanovic and B. Dutertre, "Property-directed k-induction," in *Formal Methods in Computer-Aided Design.*, 2016, pp. 86–92.

[16] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*, 2003, pp. 502–518.

[17] A. Griggio and M. Roveri, "Comparing different variants of the IC3 algorithm for hardware model checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 1026–1039, June 2016.

[18] Y. Vizel and A. Gurfinkel, "Interpolating property directed reachability," *Computer Aided Verification: 26th International Conference, CAV 2014*, pp. 260–276, 2014.