Property-Directed k-Induction

Dejan Jovanović
SRI International
dejan.jovanovic@sri.com

Bruno Dutertre
SRI International
bruno.dutertre@sri.com

***Abstract*—IC3 and k-induction are commonly used in automated analysis of infinite-state systems. We present a reformulation of IC3 that separates reachability checking from induction reasoning. This makes the algorithm more modular, and allows us to integrate IC3 and k-induction. We call this new method property-directed k-induction (PD-KIND). We show that k-induction is more powerful than regular induction, and that, modulo assumptions on the interpolation method, PD-KIND is more powerful than k-induction. Moreover, with k-induction as the invariant generation back-end of IC3, the new method can produce more concise invariants. We have implemented the method in the SALLY model checker. We present empirical results to support its effectiveness.**

I. Introduction

IC3 and k-induction are two commonly used methods in automated analysis of infinite-state systems. IC3 was originally developed for finite systems [7], [20], but it was soon adapted to the infinite-state case by relying on SMT solvers as reasoning engines [22], [9], [24], [10]. These IC3 variants have been successfully used in analysis of software. Similarly, k-induction was initially introduced in the finite-state setting [31] and was then extended to infinite-state systems [15], with similar success in software verification [16].

At its core, IC3 is based on induction. To show that a property is invariant, IC3 tries to incrementally construct an inductive strengthening of the property. Surprisingly, the relative power of k-induction and induction-based methods, with respect to the capability to construct a strengthening, has not been studied in detail.[1] It is folklore knowledge that k-induction can be stronger than induction, but this has, to the best of our knowledge, never been formally accounted for. In this paper, we show that k-induction can be strictly more powerful than regular induction, making a case for an IC3-style method that is based on k-induction. The additional reasoning power is particularly important when one works within an expressive logical theory such as the theory of arrays [21], [23]).

Although additional deductive power is desirable, we are also concerned with practical effectiveness. With this in mind, we start with IC3, a practically effective algorithm, and break it into its constituents: satisfiability checking, reachability checking, and generation of inductive invariants. Isolating these functionally independent modules allows us to replace the inductive core with k-induction, producing a method that is a natural combination of IC3 and k-induction. This method is effective in practice, and can be shown to be at least as powerful as k-induction, provided the interpolation procedure satisfies a natural property that we call finite-covering.

To summarize, the main contributions of the paper are as follows. We show that k-induction can be more powerful, and more concise, than regular induction (Section III). We decompose IC3 into functionally relevant parts (Section IV) and adopt k-induction as the core reasoning step (Section IV-C). We isolate a fundamental property of interpolation (Section IV-A) that allows us to prove the new method more powerful than k-induction. We provide experimental evidence that the new method is effective in practice (Section V). [2]

II. Background

We assume a finite set of typed variables $\vec{x}$ called *state variables*. To each variable $x \in \vec{x}$, we associate its primed version x' of the same type. We call any quantifier-free formula $F(\vec{x})$ over the state variables a *state formula*, and any quantifier-free formula $T(\vec{x}, \vec{x}')$ a *state-transition formula*. A *state* s is a type-consistent interpretation of $\vec{x}$ that assigns to each variable $x \in \vec{x}$ a value $s(x)$ over its domain. A state formula $F(\vec{x})$ holds in a state s (written $s \vDash F$) if the formula evaluates to true under the state's assignment.

A *state-transition system* is a pair $\mathfrak{S} = \langle I, T \rangle$, where $I(\vec{x})$ is a state formula describing the initial states and $T(\vec{x}, \vec{x}')$ is a state-transition formula describing the system's evolution. A state s' is a successor of a state s in $\mathfrak{S}$ if the formula $T(\vec{x}, \vec{x}')$ evaluates to true when we interpret each $x \in \vec{x}$ as $s(x)$ and each $x' \in \vec{x}'$ as $s'(x)$. A state s is *k-reachable* if there exists a sequence of states $\sigma = \langle s_0, \dots s_k \rangle$ such that, $s = s_k$, the state s_0 satisfies I, and each s_{i+1} is a successor of s_i. We call σ a *concrete trace* of the system. We also say that a state formula F is reachable in k steps if there is a k-reachable state s such that $s \vDash F$.

Given a state formula P (*the property*), we want to determine whether all the reachable states of $\mathfrak{S}$ satisfy P. If this is the case, P is an *invariant* of $\mathfrak{S}$, which we denote by $\mathfrak{S} \vDash P$. We also write $\mathfrak{S} \vDash_a^b P$ to denote that P is true in all k-reachable states for $a \leq k \leq b$. If P is not invariant, there is a concrete trace, called a *counter-example*, that reaches $\neg P$.

The research presented in this paper has been supported by NASA Cooperative Agreements NNX14AI05A and NNA10DE73C, by DARPA under agreement number FA8750-16-C-0043, and by NSF grant 1528153.

[1]Some IC3 variants, such as PDR [22], are not guaranteed to terminate even if the property is already inductive.

[2]Due to space limitations proofs we omit the proofs in this paper. The full paper with proofs and additional experimental data is available from the authors as a technical report.

ISBN: 978-0-9835678-6-8. Copyright owned jointly by the authors and FMCAD, Inc.

Definition II.1 ($\mathcal{F}$-Induction)**.** *Given a set $\mathcal{F}$ of state formulas such that $P \in \mathcal{F}$, P is* $\mathcal{F}$-inductive[3] *with respect to $\mathfrak{S}$ if*

$$I(\vec{x}) \Rightarrow \mathcal{F}(\vec{x}) \ , \qquad \text{(init)}$$
$$\mathcal{F}(\vec{x}) \wedge T(\vec{x}, \vec{x}') \Rightarrow P(\vec{x}') \ . \qquad \text{(cons)}$$

If $\mathcal{F} = \{P\}$, we say that P is inductive.

If P is inductive then it is also invariant. Since invariants are in general not inductive, a common approach to prove that P is invariant is to find a *strengthening* of P. Such a strengthening is a set of formulas $\mathcal{F}$ such that $P \in \mathcal{F}$ and $\mathcal{F}$ is inductive. If such a strengthening exists, then P is invariant.

Definition II.2 ($\mathcal{F}^k$-Induction)**.** *Given a set $\mathcal{F}$ of state formulas such that $P \in \mathcal{F}$, P is* $\mathcal{F}^k$-inductive *with respect to $\mathfrak{S}$ if*

$$I(\vec{x_0}) \wedge \bigwedge_{i=0}^{l-1} T(\vec{x}_i, \vec{x}_{i+1}) \Rightarrow \mathcal{F}(\vec{x}_l) \ , \textit{ for } 0 \leq l < k \ , \qquad \text{(k-init)}$$
$$\bigwedge_{i=0}^{k-1} (\mathcal{F}(\vec{x}_i) \wedge T(\vec{x}_i, \vec{x}_{i+1})) \Rightarrow P(\vec{x}_k) \ . \qquad \text{(k-cons)}$$

*When $\mathcal{F} = \{P\}$, we say that P is k-*inductive.

A property that is inductive is 1-inductive by definition. It is also k-inductive for any k. In the other direction, if a property P is k-inductive and the logical theory underlying the system admits quantifier elimination, then we can construct an inductive strengthening of P by eliminating quantifiers.[4] For such theories, induction and k-induction have the same deductive power but k-induction may give more succinct strengthenings. If the base theory does not admit quantifier elimination then k-induction can be more powerful than induction.[5]

III. Relative Power of Induction and k-Induction

We present examples that illustrate the deductive power of k-induction in relation to induction. To simplify the presentation, we describe transition systems as programs. We use the quantifier-free fragment of the extensional theory of arrays [27], denoted by $\mathcal{T}_{\mathbf{arr}}$, and an extension of $\mathcal{T}_{\mathbf{arr}}$ with array constants [13], denoted by $\mathcal{T}^{\mathbf{c}}_{\mathbf{arr}}$. The theory $\mathcal{T}_{\mathbf{arr}}$ is axiomatized as $\text{WRITE}(a, i, v)[i] = v$, $i \neq j \Rightarrow \text{WRITE}(a, i, v)[j] = a[j]$, and $a[i] \neq b[i] \Rightarrow a \neq b$. The extended theory $\mathcal{T}^{\mathbf{c}}_{\mathbf{arr}}$ is obtained by adding a construct $\mathbf{c}(v)$ that represents the constant array with value v, and the axiom $\mathbf{c}(v)[i] = v$.

The quantifier-free fragments of both theories are decidable [32], [13] and are very useful in practice. For example, one can model integer sets in the theory $\mathcal{T}^{\mathbf{c}}_{\mathbf{arr}}$ as arrays that map integers to Booleans, and define set operations as $x \in a \equiv a[x]$, $\emptyset \equiv \mathbf{c}(\mathbf{false})$, $a \cup \{x\} \equiv \text{WRITE}(a, x, \mathbf{true})$.

[3]This is the same idea as induction relative to $\mathcal{F}$ used in [7].

[4]If P is k-inductive then $P \wedge \circ P \wedge \circ \circ P \wedge \ldots \wedge \overbrace{\circ \circ \cdots \circ}^{k-1} P$ is inductive, where $\circ$ stands for "next state".

[5]We postulate that, for theories such as pure Boolean logic or linear arithmetic, k-induction is exponentially more succinct than induction. Proving this postulate would entail new complexity results on quantifier elimination (e.g., [33]) and would require a much more sophisticated analysis.

```
1  int i, j;
2  map<int,int> a; // int -> int
3
4  // I: write 0 at a[0]
5  i = j = 0;
6  a[0] = 0;
7
8  // T: write 0 at a[i] from a[j]
9  while (true) {
10   j = rand() % (i+1);
11   i = (i+1) % N;
12   a[i] = a[j];
13 }
```

Fig. 1. Writing 0 to array a in a circular fashion, resetting every N steps.

Although their quantifier-free fragments are decidable, the full array theories are not decidable [8] and therefore do not admit quantifier elimination.[6] This makes them good candidates for relating the powers of induction and k-induction.

Example III.1. *The program in Figure 1 sets the elements $a[0], \ldots, a[N-1]$ to 0 in a circular fashion. This program can be encoded as a transition system in theory $\mathcal{T}_{\mathbf{arr}}$, with initial states defined by lines 5-6, and transition relation defined by lines 10-12. Now, consider the property $P \equiv (a[0] = 0)$. This property is clearly an invariant of the system. One can show that it is $(N+1)$-inductive: Any sequence of $N+1$ states must include a transition that resets i to 0. From then on, all transitions increment i to an integer no more than $N-1$, pick a j between 0 and i, and copy $a[j]$ into $a[i]$. If P holds at all states in this sequence, and if $i = N-1$ in the last state of this sequence, then $a[0], \ldots, a[N-1]$ are all 0 in this state. If $i \neq N-1$ in the last state then the next transition keeps $a[0]$ unchanged. These observations are enough to conclude that P is $(N+1)$-inductive. On the other hand, P is not k-inductive for any $k \leq N$. Moreover,* any inductive strengthening of P must have size at least N, *such as for example $P \wedge \bigwedge_{k=1}^{N-1} (a[k] = 0 \vee i < k)$.*

Lemma III.1. *There exists a sequence of state-transition systems $\mathfrak{S}_N$ and a property P, such that for any N the property P is N-inductive in $\mathfrak{S}_N$, but the shortest inductive strengthening of P has a size larger than N.*

The relationship between induction and k-induction is explored in [5], where the authors show that induction is as powerful as k-induction for theories that admit "feasible interpolation." Feasible interpolation ensures that the inductive strengthening is polynomial in the size of the proof. This is in line with Lemma III.1, since the proof itself can be exponential, resulting in potential blowup of the invariant.

Example III.2. *Consider the program in Figure 2. This program sets the elements $a[0], \ldots, a[i], \ldots$ to 0 in rounds of N steps. Variable c counts the number of steps in the current round and variable J stores the indices of the elements of a that have been written to. The program can be encoded as a transition system in theory $\mathcal{T}^{\mathbf{c}}_{\mathbf{arr}}$, with initial states defined*

[6]For example, $\exists i \ . \ a[i] = 0$ does not have a quantifier-free equivalent.

ISBN: 978-0-9835678-6-8. Copyright owned jointly by the authors and FMCAD, Inc.

```
 1  int c, i, j;
 2  set<int> J;       // int -> bool
 3  map<int, int> a; // int -> int
 4
 5  // I: write 0 at a[0]
 6  c = 0;
 7  a[0] = 0;
 8  J.insert(0)
 9
10  // T: write 0 at a[i] from a[j]
11  while (true) {
12    c = (c+1) % N;
13    if (c == 0) {
14      J.clear();
15      i = 0;
16      a[i] = 0;
17    } else {
18      i = rand();
19      j = J.pick_rand();
20      a[i] = a[j];
21    }
22    J.insert(i)
23  }
```

Fig. 2. Writing 0 to array a while keeping a set J of written-to indices, resetting every N steps.

by lines 6-8, and transition relation expressed by lines 12-22. As previously, let the property be $P \equiv (a[0] = 0)$ then P is invariant. By the pigeon-hole principle, P is $(N+1)$-inductive: Any sequence of $N+1$ states must include a "reset" that sets $i = 0$ and $J = \{0\}$. From this reset point, all transitions sets some $a[i]$ to 0 and adds i to J. However, there is no inductive strengthening P' of P. *For P' to be inductive, it would need to ensure that for all $j \in J$ we have $a[j] = 0$ but this can not be expressed in the quantifier-free fragment of theory $\mathcal{T}^{c}_{arr}$.*

Lemma III.2. *There exists a state-transition system $\mathfrak{S}$ and a property P, such that the property P is k-inductive for $k > 1$ but there is no inductive strengthening of P.*

The additional power of k-induction comes with a computational price: checking whether a property is k-inductive requires $k+1$ satisfiability checks and a potentially expensive unrolling of the transition relation. The method we propose in this paper alleviates this issue by using an incremental approach that minimizes the need for unrolling.

IV. Algorithm

We reason about transition systems in the satisfiability modulo theories (SMT) framework [2]. Specifically, we assume that the transition system is described in a theory where quantifier-free satisfiability is decidable.

A. Satisfiability Checking

Given a state formula F, we denote with $T[F]^k$ the unrolling of T to length k where F holds in the intermediate states. For $k > 1$, $T[F]^k(\vec{x}, \vec{x}')$ is then defined as

$$T(\vec{x}, \vec{w}_1) \wedge \bigwedge_{i=1}^{k-1} (F(\vec{w}_i) \wedge T(\vec{w}_i, \vec{w}_{i+1})) \wedge T(\vec{w}_{k-1}, \vec{x}')$$

where $\vec{w}$ are the state variables in the intermediate states. For $k = 0$ and $k = 1$, we set $T^0[F](\vec{x}, \vec{x}') \equiv (\vec{x} = \vec{x}')$ and $T^1[F](\vec{x}, \vec{x}') \equiv T(\vec{x}, \vec{x}')$. When $F \equiv \mathbf{true}$, we omit it and simply write T^k.

A basic step in our algorithms is to check the satisfiability of formulas of the form

$$A(\vec{x}) \wedge T[B]^k(\vec{x}, \vec{w}, \vec{y}) \wedge C(\vec{y}) \ , \qquad (1)$$

where A, B, and C are state formulas. We denote by CHECK-SAT$(A, T[B]^k, C)$ an (SMT-based) procedure that checks satisfiability of formula (1), and returns a model if the formula is satisfiable. In addition, we require two artifacts from the SMT solver: *interpolants* and *generalizations*.

Definition IV.1 (Interpolant)**.** *If the formula* (1) *is unsatisfiable, a formula $J(\vec{y})$ is a* state interpolant *if*

1) *$A(\vec{x}) \wedge T[B]^k(\vec{x}, \vec{w}, \vec{y}) \Rightarrow J(\vec{y})$, and*
2) *$J(\vec{y})$ and $C(\vec{y})$ are inconsistent.*

Definition IV.2 (Generalization)**.** *If the formula* (1) *is satisfiable, we call a formula $G(\vec{x})$ a* state generalization *if*

1) *$A(\vec{x})$ and $G(\vec{x})$ are consistent, and*
2) *$G(\vec{x}) \Rightarrow \exists \vec{w}, \vec{y} \,.\, T[B]^k(\vec{x}, \vec{w}, \vec{y}) \wedge C(\vec{y})$.*

Interpolation provides forward learning. An interpolant over-approximates the set of states reachable from A via $T[B]^k$ and is enough to refute C. Generalization is the dual and provides backward learning. A generalization G is consistent with A and under-approximates the set of states that can reach C via $T[B]^k$.

Our notion of a state interpolant is more specific than the one usually considered in general interpolation, and our definition can be easily satisfied: formula $\neg C$ is always an interpolant (so interpolants exists in our case even if the underlying theory does not support general interpolation). Similarly, one can construct a generalization G from a model v of formula (1) by substitution (i.e., the formula $(A \wedge T[B]^k)[\vec{w}/v(\vec{w}), \vec{y}/v(\vec{y})]$ is a trivial generalization). Although correct, trivial interpolants and generalizations are not ideal for practical applications. In particular, they do not satisfy the following property.

Definition IV.3 (Finite Cover Property)**.** *An interpolation (resp. generalization) procedure has the* finite cover property *(is finite-covering) when, for a fixed $T[B]^k$ and A (resp C), it can only produce a finite number of distinct interpolants (resp. generalizations).*

Interpolation is a well-studied topic [28], [12] and it is available in several SMT solvers. Effective generalization in SMT was introduced in [24] (as model-based projection) for specific use in a PDR engine. There are known generalization methods for the theories of linear arithmetic [24], arrays [23], and algebraic data-types [6]. These methods have the finite cover property. On the other hand, most interpolation procedures are proof-based and do not ensure finite covering.

Both interpolation and generalization approximate quantifier elimination. For theories that admit quantifier elimination, one can construct precise interpolants by eliminating $\vec{x}$ and $\vec{w}$ from $A \wedge T[B]^k$ and precise generalizations by eliminating $\vec{w}$ and

$\vec{y}$ from $T[B]^k \wedge B$. Such precise procedures have the finite cover property, and it is not unreasonable to expect the same from interpolation and generalization. This is the case for pure SAT problems. In SAT, interpolants can always be expressed as clauses, while generalizations can be expressed as prime implicants, both of which guarantee the finite cover property.[7] The finite-cover property for interpolants is an incremental form of the related notion of uniform interpolation [30]: uniform interpolation requires a single interpolant instead of a finite set.

Example IV.1. *Consider the system $\mathfrak{S} = \langle I, T\rangle$ defined as $I \equiv (x = 0)$, $T \equiv (x' = x + 1)$ where x is a real-valued variable. Let P be the formula $0 \le x \vee x \ge 1$. To check whether P is inductive, we can ask the following satisfiability query to the SMT solver*

$$\overbrace{(0 \le x \vee x \ge 1)}^{A_1} \wedge \overbrace{(x' = x + 1)}^{T^1} \wedge \overbrace{\neg(0 \le x' \vee x' \ge 1)}^{B_1 \equiv \neg A_1} \ .$$

This formula is satisfiable in a model $x \mapsto -0.5$, $x' \mapsto 0.5$. We can generalize this model to $G \equiv (-1 < x < 0)$; any state that satisfies G is a counterexample to induction of P. We can then check whether G intersects with the initial states and whether G is reachable in one step, by making two separate queries

$$\overbrace{(x = 0)}^{A_2} \wedge \overbrace{(x' = x)}^{T^0} \wedge \overbrace{(-1 < x' < 0)}^{B_2} \ ,$$
$$\underbrace{(x = 0)}_{A_3} \wedge \underbrace{(x' = x + 1)}_{T^1} \wedge \underbrace{(-1 < x' < 0)}_{B_3} \ .$$

Both queries are unsatisfiable. From the first query, we can get an interpolant $J_0(x') \equiv (x' \ge 0)$ that refutes G in the initial states. From the second query, we can get an interpolant $J_1(x') \equiv (x' \ge 1)$ that refutes G after one transition. Although P itself is not inductive, the two interpolants give us a strengthening: the formula $P' \equiv P \wedge (J_0(x) \vee J_1(x))$ is inductive.

B. Reachability

Problem IV.1 (k-reachability)**.** *Given a state formula F that is not reachable in fewer then k steps, check whether F is reachable in k steps.*

The reachability problem can be solved by bounded model checking [3], but we discuss an alternative method that does not require unrolling the transition relation. We introduce the concept of k-interpolation as a way to learn from failures of k-reachability.

Definition IV.4 (k-interpolant)**.** *Given a system $\mathfrak{S}$ and state formula F that is unreachable in $\le k$ steps (system is k-inconsistent with F), a state formula J is a state k-interpolant for F if*

$$\mathfrak{S} \models_0^k J \ , \qquad J \text{ and } F \text{ are inconsistent} \ .$$

[7]For arithmetic theories, finite-covering interpolants can be generated using model-based procedures such as MCSat [14].

As with regular interpolation, the formula $\neg F$ itself is a trivial k-interpolant. We can also construct a k-interpolant by calling a standard interpolation procedure $k + 1$ times: If $\mathfrak{S}$ and F are k-inconsistent, then $I(\vec{x}) \wedge T^i(\vec{x}, \vec{w}, \vec{x}') \wedge F(\vec{x}')$ is unsatisfiable for $0 \le i \le k$. From these inconsistencies we can obtain interpolants $J_0,\ldots, J_k$, and the formula $J \equiv (J_0 \vee \ldots \vee J_k)$ is a k-interpolant for F. Moreover, if the interpolation procedure has the finite-cover property then so does the k-interpolation procedure.

To check k-reachability, we adopt the incremental depth-first reachability method of IC3, which relies on local reasoning. The procedure maintains a sequence $\mathcal{R}_0$, $\mathcal{R}_1$, $\ldots$of *reachability frames*. Frame $\mathcal{R}_i$ is a set of state formulas that over-approximates the set of states reachable in i steps or less. This implies that $\mathfrak{S} \models_0^i \mathcal{R}_i$. Unlike IC3/PDR, we *do not require the frames to be monotonic*; we may have $\mathcal{R}_{i+1} \not\subseteq \mathcal{R}_i$.[8]

This setup allows us to build k-interpolants efficiently provided an extra local condition holds. If $k = 0$, we just take the interpolant of $I \wedge T^0 \wedge F$. If $k > 0$ and the formula F is not reachable in up to k steps, and if, in addition, F is not reachable in one step from $\mathcal{R}_{k-1}$, then both $I \wedge T^0 \wedge F$ and $\mathcal{R}_{k-1} \wedge T \wedge F$ are inconsistent. We can then obtain interpolants J_1 and J_2 for these two formulas and $(J_1 \vee J_2)$ is a k-interpolant for F. This k-interpolant, which we denote by EXPLAIN$(\mathfrak{S}, k, F)$, is potentially more concise than the one described before and it is obtained by local reasoning only. Although EXPLAIN has an additional precondition, our algorithm ensures that this holds whenever EXPLAIN is called.

Lemma IV.1. *Starting from a fixed finite frame sequence $\mathcal{R}_0, \mathcal{R}_1, \ldots$, if the only formulas we add to the frames are obtained through the* EXPLAIN *procedure, and the interpolation procedure is finite-covering, then the* EXPLAIN *procedure is also finite-covering.*

Algorithm 1 Check k-reachability of F.

```
Require: $\mathfrak{S} \models_0^i \mathcal{R}_i$ for $0 \le i \le k$, $\mathfrak{S} \models_0^{k-1} \neg F$.
Ensure: $\mathfrak{S} \models_0^i \mathcal{R}_i$ for $0 \le i \le k$. If not reachable, $\mathcal{R}_{k-1} \wedge T \wedge F$
    is unsatisfiable.
function REACHABLE($\mathfrak{S}$, $k$, $F$)
    if $k = 0$ then return CHECK-SAT($I, T^0, F$)
    loop
        if CHECK-SAT($\mathcal{R}_{k-1}, T, F$) then
            $G \leftarrow$ GENERALIZE($\mathcal{R}_{k-1}, T, F$)
            if REACHABLE($\mathfrak{S}, k-1, G$) then
                return true
            else
                $E \leftarrow$ EXPLAIN($\mathfrak{S}, k-1, G$)
                $\mathcal{R}_{k-1} \leftarrow \mathcal{R}_{k-1} \cup \{E\}$
        else return false
```

Finally, our reachability routine REACHABLE$(\mathfrak{S}, k, F)$ performs a step-wise search for a concrete trace by using a depth-first search strategy. It tries to reach the initial states

[8]From an implementation perspective, this gives flexibility in garbage collection. We can remove any subset of formulas from any frame $\mathcal{R}_i$ without compromising correctness.

backwards. To reach F at frame k, we check first whether F can be reached in one transition from the previous frame $\mathcal{R}_{k-1}$. If no such transition is possible, then F is not reachable. Otherwise, we get a state s that satisfies $\mathcal{R}_{k-1}$ and from which F is reachable in one step. The generalization procedure gives us a formula G that generalizes s: every state that satisfies G has a successor that satisfies F. We then recursively check whether G is reachable. The recursive call will either find a path from the initial states to G, in which case F is reachable, or determine that G is not reachable, in which case we can learn an explanation E of the reachability failure and eliminate G. Learning E eliminates G as a potential step backward, and we continue.

Lemma IV.2. *Algorithm 1 solves the k-reachability problem. If F is not reachable then, upon completion, either $k = 0$ and F is inconsistent with I, or $k > 0$ and F is not reachable in one step from $\mathcal{R}_{k-1}$. In addition, if the interpolation or the generalization procedure is finite-covering, then the procedure always terminates.*

We use a variant of the REACHABLE procedure to check whether F is reachable in steps k_1 to k_2. We denote this by $(r, l) = \text{REACHABLE}(\mathfrak{S}, k_1, k_2, F)$. This extension of the REACHABLE procedure is a straightforward loop from k_1 to k_2, and has the same precondition as the single check version (on k_1). In the return value, r denotes the reachability result (true/false), and, if r is true, l is the length of the shortest trace that can reach F. The postcondition (and hence Lemma IV.2) of the iterative extension is also the same (on k_2).

C. Property-Directed k-Induction

We now present the main procedure of PD-KIND. This procedure checks whether a property P is invariant for a system $\mathfrak{S} = \langle I, T\rangle$. It does so by iteratively trying to construct a k-inductive strengthening of P for some $k > 0$. The overall idea behind the procedure is simple. Assume a set of formulas $\mathcal{F}_{\text{ABS}}$ that is a strengthening of P and is valid in $\mathfrak{S}$ for up to n steps. In other words, $\mathcal{F}_{\text{ABS}}$ is an over-approximation of states reachable in n steps or less. Then, the set $\mathcal{F}_{\text{ABS}}$ satisfies (k-init) for all $1 \le k \le n+1$. We can pick any such k and try to show that $\mathcal{F}_{\text{ABS}}$ is k-inductive by checking whether it also satisfies (k-cons). Each iteration of the procedure PD-KIND does this check. The core of our algorithm is procedure PUSH that either finds a counter-example to P or produces a new strengthening $\mathcal{G}_{\text{ABS}} \subseteq \mathcal{F}_{\text{ABS}}$. This new set $\mathcal{G}_{\text{ABS}}$ satisfies (k-cons) with respect to $\mathcal{F}_{\text{ABS}}$. The set $\mathcal{G}_{\text{ABS}}$ is then $\mathcal{F}^k_{\text{ABS}}$-inductive. If $\mathcal{G}_{\text{ABS}} = \mathcal{F}_{\text{ABS}}$, we can conclude that P is invariant. Otherwise, we know that $\mathcal{G}_{\text{ABS}}$ is valid (at least) up to index $n + 1$. Procedure PUSH actually returns an integer n_p such that $\mathcal{G}_{\text{ABS}}$ is valid up to n_p. This index n_p is the length of the shortest trace of $\mathfrak{S}$ that reaches $\neg\mathcal{F}_{\text{ABS}}$; it is guaranteed to be at least $n+1$ but it may be larger. At this point, we repeat the loop with $\mathcal{G}_{\text{ABS}}$ as our current strengthening and n_p as our new index.

In addition to the set of formulas $\mathcal{F}_{\text{ABS}}$, procedure PD-KIND associates with each each $F_{\text{ABS}} \in \mathcal{F}_{\text{ABS}}$ information about a potential counter-example to P that the formula F_{ABS}

Algorithm 2 Main PD-KIND procedure.

```
Require: $\mathfrak{S} = \langle I, T\rangle$ and $I \Rightarrow P$
1  function PD-KIND($\mathfrak{S}, P$)
2      $n \leftarrow 0$
3      $\mathcal{F} \leftarrow \{(P, \neg P)\}$
4      loop
5          pick $k$-induction depth $1 \le k \le n + 1$
6          $\langle \mathcal{F}, \mathcal{G}, n_p\rangle \leftarrow$ PUSH($\mathfrak{S}, \mathcal{F}, P, n, k$)
7          if $P$ marked invalid then return invalid
8          if $\mathcal{F} = \mathcal{G}$ then return valid
9          $n \leftarrow n_p$
10         $\mathcal{F} \leftarrow \mathcal{G}$
```

eliminates. The set $\mathcal{F}_{\text{ABS}}$ and this additional information is represented in the form of an induction frame. Let $\mathbb{F}$ denote the set of all state formulas in our theory.

Definition IV.5 (Induction Frame)**.** *A set of tuples $\mathcal{F} \subset \mathbb{F} \times \mathbb{F}$ is an* induction frame *at index n if $(P, \neg P) \in \mathcal{F}$ and the following holds for all $(F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}$:*

1) *F_{ABS} is valid up to n steps and refutes F_{CEX}, and*
2) *F_{CEX}-states can be extended to a counter-example to P.*

If $I \Rightarrow P$, then the set $\mathcal{F} = \{(P, \neg P)\}$ is an induction frame at index 0. Given an induction frame $\mathcal{F}$, we denote by $\mathcal{F}_{\text{ABS}}$ the strengthening represented by $\mathcal{F}$, i.e., $\mathcal{F}_{\text{ABS}} = \{F_{\text{ABS}} \mid (F_{\text{ABS}}, F_{\text{CEX}}) \in \mathcal{F}\}$. With this in mind, the procedure PD-KIND is presented in Algorithm 2.

D. The PUSH *Procedure*

The core of the PD-KIND algorithm is the PUSH procedure (Algorithm 3). This procedure takes as input an induction frame $\mathcal{F}$ at index n, and tries to push formulas of the frame using k-induction where $1 \le k \le n + 1$. Figure 3 illustrates the formulas and frame indices over which PUSH operates.

Since $\mathcal{F}$ is an induction frame at n, we know that $\mathcal{F}_{\text{ABS}}$ is valid up to index n. In each iteration, the procedure picks one yet unprocessed $(F_{\text{ABS}}, F_{\text{CEX}})$ from $\mathcal{F}$. Both F_{ABS} and $\neg F_{\text{CEX}}$ hold up to index n in $\mathfrak{S}$.

First, the procedure checks whether F_{ABS} is $\mathcal{F}^k_{\text{ABS}}$-inductive (lines 9-12). If so, then we know that F_{ABS} is valid at least up to position $n + 1$. We call this a successful push and we add $(F_{\text{ABS}}, F_{\text{CEX}})$ to the set of pushed obligations $\mathcal{G}$, and continue with the next obligation. If the k-induction check fails, then we have a model (counterexample to induction) m_{CTI}. This is a trace of length $k + 1$ in which $\mathcal{F}_{\text{ABS}}$ holds for the first k states but F_{ABS} is false in the last state.

The procedure does not use m_{CTI} yet. Instead, it checks whether the counterexample formula F_{CEX} is reachable from $\mathcal{F}_{\text{ABS}}$ (lines 15-24). If the query at line 15 is satisfiable, it has a model m_{CEX}. Like m_{CTI}, this model is a trace of length $k + 1$; it starts with k states that satisfy $\mathcal{F}_{\text{ABS}}$ and ends with a state that satisfies F_{CEX} (thus, from the first state of m_{CEX} we can reach $\neg P$). At this point, we generalize m_{CEX} to a formula G_{CEX}. From any state that satisfies G_{CEX}, one can reach $\neg P$. Formula G_{CEX} is then a potential counterexample for P. We check whether G_{CEX} is reachable from the initial

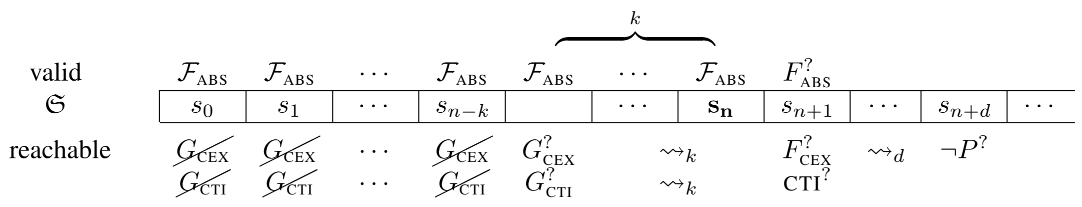
k
valid
$\mathcal{F}_{\text{ABS}}$ $\mathcal{F}_{\text{ABS}}$ $\cdots$ $\mathcal{F}_{\text{ABS}}$ $\mathcal{F}_{\text{ABS}}$ $\cdots$ $\mathcal{F}_{\text{ABS}}$ $F^?_{\text{ABS}}$
$\mathfrak{S}$
s_0 s_1 $\cdots$ s_{n-k} $\cdots$ $\mathbf{s_n}$ s_{n+1} $\cdots$ s_{n+d} $\cdots$
reachable
$\cancel{G_{\text{CEX}}}$ $\cancel{G_{\text{CEX}}}$ $\cdots$ $\cancel{G_{\text{CEX}}}$ $G^?_{\text{CEX}}$ $\rightsquigarrow_k$ $F^?_{\text{CEX}}$ $\rightsquigarrow_d$ $\neg P^?$
$\cancel{G_{\text{CTI}}}$ $\cancel{G_{\text{CTI}}}$ $\cdots$ $\cancel{G_{\text{CTI}}}$ $G^?_{\text{CTI}}$ $\rightsquigarrow_k$ CTI?

Fig. 3. Illustration of the formulas and frame indices over which PUSH operates.

states of $\mathfrak{S}$. Because we know that F_{CEX} is not n-reachable, G_{CEX} can't be reached in less than $n-k+1$ steps. So we check reachability of G_{CEX} at positions $n-k+1 \ldots n$. If G_{CEX} is reachable, then so is $\neg P$ and we mark P as invalid. Otherwise, we call the EXPLAIN procedure, which returns a new fact G_{ABS} that eliminates G_{CEX}. The new fact G_{ABS} is true up to position n, and refutes G_{CEX}, so we can add the new induction obligation $(G_{\text{ABS}}, G_{\text{CEX}})$ to $\mathcal{F}$, strengthening $\mathcal{F}$, and try again with a potential counter-example eliminated.

In the remaining case, we have a counterexample m_{CTI} to the k-inductiveness of F_{ABS}. Since the query at line 15 is not satisfiable and $\mathcal{F}_{\text{ABS}} \Rightarrow \neg F_{\text{CEX}}$, we know that $\neg F_{\text{CEX}}$ is $\mathcal{F}^k_{\text{ABS}}$ inductive. We first apply generalization to m_{CTI} to construct a formula G_{CTI}. From any state that satisfies G_{CTI}, we can reach $\neg F_{\text{ABS}}$ in k steps. If G_{CTI} is reachable in $\mathfrak{S}$ then $\neg F_{\text{ABS}}$ is also reachable, so F_{ABS} can't be part of a valid strengthening of P. This check is performed at line 28; as previously, it is enough to check reachability of G_{CTI} at positions $n-k+1, \ldots, n$. If G_{CTI} is reachable, we can't push F_{ABS}. Instead, we replace the triple $(F_{\text{ABS}}, F_{\text{CEX}})$ by the weaker obligation $(\neg F_{\text{CEX}}, F_{\text{CEX}})$. This new obligation can be immediately pushed to $\mathcal{G}$. On the other hand, if G_{CTI} is not reachable then we strengthen F_{ABS} with a new fact G_{ABS} learned from procedure EXPLAIN. This eliminates the counterexample to k-induction and the procedure continues.

At lines 30-31 of the procedure, we know that $\neg F_{\text{ABS}}$ is reachable in $\mathfrak{S}$ and that this requires at least $n+1$ transitions. It is useful to make this more precise by computing the actual length of the shortest path to $\neg F_{\text{ABS}}$ (line 30). This length is stored in variable n_p (if it's smaller than n_p's current value).

After the loop terminates, PUSH returns the set of successfully pushed induction obligations $\mathcal{G}$, the modified set $\mathcal{F}$ of k-induction assumptions for $\mathcal{G}$, and the shortest refutation length n_p for any $F_{\text{ABS}} \in \mathcal{F}_{\text{ABS}}$ that was not successfully pushed. The procedure does not only add to the original set $\mathcal{F}$, it also actively modifies it (line 37). Unlike existing IC3-based procedures where frames are explored in succession, keeping track of n_p allows us to perform "jumps" that move to deeper frames faster. This is because $\mathcal{F}_{\text{ABS}}$ is valid up to position $n_p - 1 \geq n$, and the facts in $\mathcal{G}_{\text{ABS}}$ are valid up to position $n_p \geq n+1$.[9]

Assuming that PD-KIND terminates, it is not hard to show that it returns the correct result. If PD-KIND terminates with P marked invalid, then we have found a counter-example

Algorithm 3 Push $\mathcal{F}$ with k-induction.

Require: $\mathcal{F}$ is a valid frame for P at position n, $1 \leq k \leq n+1$, $(P, \neg P) \in \mathcal{F}$.
Ensure: $\mathcal{F}$ is a valid frame for P at position $n_p - 1 \geq n$, $\mathcal{G} \subseteq \mathcal{F}$ is $\mathcal{F}^k$-inductive, and P marked invalid or $(P, \neg P, 0) \in \mathcal{F}_p$.

```
1  function PUSH($\mathfrak{S}$, $\mathcal{F}$, $P$, $n$, $k$)
2      push elements of $\mathcal{F}$ to $\mathcal{Q}$          ▷ $\mathcal{Q}$ is a priority queue.
3      $\mathcal{G} \leftarrow \{\}$          ▷ Pushed facts, i.e. $\mathcal{G}_{\text{ABS}}$ is $\mathcal{F}^k_{\text{ABS}}$-inductive.
4      $n_p \leftarrow n + k$          ▷ Keeps track of the shortest CTI position.
5      while $P$ not marked invalid, $\mathcal{Q}$ not empty do
6          pop $(F_{\text{ABS}}, F_{\text{CEX}})$ from $\mathcal{Q}$
7
8          ____________          ▷ Is $F_{\text{ABS}}$ $\mathcal{F}^k_{\text{ABS}}$-inductive?
9          $(sat_{\text{CTI}}, m_{\text{CTI}}) \leftarrow$ CHECK-SAT$(\mathcal{F}_{\text{ABS}}, T[\mathcal{F}_{\text{ABS}}]^k, \neg F_{\text{ABS}})$
10         if not $sat_{\text{CTI}}$ then
11             $\mathcal{G} \leftarrow \mathcal{G} \cup \{(F_{\text{ABS}}, F_{\text{CEX}})\}$          ▷ $\mathcal{G}_{\text{ABS}}$ is $\mathcal{F}^k_{\text{ABS}}$-inductive.
12             continue
13
14         ______________          ▷ Is $F_{\text{CEX}}$ reachable?
15         $(sat_{\text{CEX}}, m_{\text{CEX}}) \leftarrow$ CHECK-SAT$(\mathcal{F}_{\text{ABS}}, T[\mathcal{F}_{\text{ABS}}]^k, F_{\text{CEX}})$
16         if $sat_{\text{CEX}}$ then
17             $G_{\text{CEX}} \leftarrow$ GENERALIZE$(m_{\text{CEX}}, T^k, F_{\text{CEX}})$
18             if REACHABLE$(\mathfrak{S}, n-k+1, n, G_{\text{CEX}})$ then
19                 mark $P$ invalid          ▷ $I \rightsquigarrow G_{\text{CEX}} \rightsquigarrow_k F_{\text{CEX}} \rightsquigarrow \neg P$.
20             else
21                 $G_{\text{ABS}} \leftarrow$ EXPLAIN$(\mathfrak{S}, n, G_{\text{CEX}})$
22                 $\mathcal{F} \leftarrow \mathcal{F} \cup \{(G_{\text{ABS}}, G_{\text{CEX}})\}$          ▷ Eliminate CEX.
23                 push $(G_{\text{ABS}}, G_{\text{CEX}})$, $(F_{\text{ABS}}, F_{\text{CEX}})$ to $\mathcal{Q}$
24             continue
25
26         __________          ▷ Analyze the induction failure.
27         $G_{\text{CTI}} \leftarrow$ GENERALIZE$(m_{\text{CTI}}, T^k, \neg F_{\text{ABS}})$
28         $(r_{\text{CTI}}, n_{\text{CTI}}) \leftarrow$ REACHABLE$(\mathfrak{S}, n-k+1, n, G_{\text{CTI}})$
29         if $r_{\text{CTI}}$ then          ▷ $I \rightsquigarrow_{n_{\text{CTI}}} G_{\text{CTI}} \rightsquigarrow_k \neg F_{\text{ABS}}$.
30             $(r_{\text{CTI}}, n_{\text{CTI}}) \leftarrow$ REACHABLE$(\mathfrak{S}, n+1, n_{\text{CTI}}+k, \neg F_{\text{ABS}})$
31             $n_p \leftarrow \min(n_p, n_{\text{CTI}})$
32             $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\neg F_{\text{CEX}}, F_{\text{CEX}})\}$
33             $\mathcal{G} \leftarrow \mathcal{G} \cup \{(\neg F_{\text{CEX}}, F_{\text{CEX}})\}$
34         else          ▷ $I \not\rightsquigarrow_{\leq n} G_{\text{CTI}} \rightsquigarrow_k \neg F_{\text{ABS}}$.
35             $G_{\text{ABS}} \leftarrow$ EXPLAIN$(\mathfrak{S}, n, G_{\text{CTI}})$          ▷ $G_{\text{ABS}} \Rightarrow \neg G_{\text{CTI}}$.
36             $G_{\text{ABS}} \leftarrow F_{\text{ABS}} \wedge G_{\text{ABS}}$          ▷ $G_{\text{ABS}} \Rightarrow \neg F_{\text{CEX}}$.
37             $\mathcal{F} \leftarrow \mathcal{F} \cup \{(G_{\text{ABS}}, F_{\text{CEX}})\} \setminus \{(F_{\text{ABS}}, F_{\text{CEX}})\}$
38             push $(G_{\text{ABS}}, F_{\text{CEX}})$ to $\mathcal{Q}$.
39
40     return $\langle \mathcal{F}, \mathcal{G}, n_p \rangle$
```

[9]We have observed significant frame jumps in practice although this is problem-specific.

to the property. On the other hand, if PD-KIND terminates when the inductive frames become equal, i.e. $\mathcal{F} = \mathcal{G}$, then $\mathcal{F}_{\text{ABS}}$ is a k-inductive strengthening of P and P is therefore valid. In general, for infinite domains, even termination of the PUSH procedure is not guaranteed. But, a finite-covering interpolation procedure ensures termination: the number of new facts that PUSH can learn is finite, and this bounds both the number of possible refinement steps, and the number of new counter-examples that can be found in line 15.

The PD-KIND procedure, as presented, has the freedom to choose the induction depth k in each iteration (line 5). We call a strategy for picking the depth increasing if it guarantees that, for every k, PD-KIND eventually picks induction depths k' larger than k.

Lemma IV.3. *If the interpolation procedure is finite-covering, then the* PUSH *procedure terminates. If the property P is k-inductive for some $k > 0$, and* PD-KIND *uses an increasing strategy for k, then the* PD-KIND *procedure terminates.*

Proof. (Sketch) If the property P is k inductive, then PD-KIND will eventually pick only depths $k' \geq k$. For any such k' no counterexamples can be found at line 15, because any m_{CEX} could be extended to a counter-example of P, violating the assumption that P is k-inductive. If no new counter-examples can be found then, as PD-KIND goes from frame to frame, the only new facts that can be added to the frame are either obtained from refinement on line 35, where existing facts are replaced with stronger facts, or line 36, where facts are weakened to a counter-example refutation. Since we know that no new counter-examples can be found, the latter can only happen a finite number of times. Therefore, the size of the frame can not increase indefinitely, and will eventually converge to a state where $\mathcal{F} = \mathcal{G}$. □

V. Experimental Evaluation

We have implemented PD-KIND in the SALLY model-checker.[10] The implementation of the procedure itself is rather small (1.2 Kloc of C++) and follows the presentation of the paper. As our back-end SMT solver we combine YICES2 [17], [18] and MATHSAT5 [11]. YICES2 is used for all satisfiability queries and for generalization, while MATHSAT5 is used for interpolation. We use two solvers because YICES2 supports model-based generalization (but not interpolation), and MATHSAT5 supports interpolation (but not generalization at the time we started implementing SALLY). This combination incurs some overhead as we solve every unsatisfiable problem twice, once with YICES2 to know that the problem is unsatisfiable and once with MATHSAT5 to get an interpolant.[11] The default strategy for picking the parameter k in PUSH is to increment by one in each iteration.

We have evaluated the new procedure on a range of existing and new benchmarks. Several of our benchmarks are related to fault-tolerant algorithms (`oral-messages` [25], `tte-synchro` and `tta-startup` [19], `unified-approx` [29], `azadmanesh-kieckhafer` [1], `approximate-agreement` [26], and `hacms` problem sets). We also used benchmarks from software model checking (`cav12` [9], `ctigar` [4]). The `lustre` benchmarks are from the benchmark suite of the KIND model-checker, and `cons` are simple concurrent programs. Some of the benchmarks were obtained from an existing repository [12]. Since our tool does not yet handle integer reasoning properly, we converted all the integer variables to the real type. All problems are flat transition systems and we translated them to the input languages of other tools in a straightforward manner. The software benchmarks come from a public repository and were already encoded in SMTLIB2 as flat transition systems.[13]

To put the results in context, we compare PD-KIND with other state-of-the-art, infinite-state model checkers, namely, Z3 [22], NUXMV [10], and SPACER [24]. The results are presented in Table 4. Each solver was run with a timeout of 20 minutes. Each column of the table corresponds to a different model-checking engine, and each row corresponds to a different problem set. For each problem set and tool combination we report the number of problems that the tool has solved, how many of the solved problems were valid and invalid properties, and the total time (in seconds) that the tool took to solve those problems.

The evaluation shows that the new method is effective and robust on real-world problems: on all problem sets, PD-KIND either solves the most problems or is very close to the best engine. PD-KIND is good at both proving properties correct and finding counter-examples. When proving invariants, PD-KIND benefits from k-induction and can in some cases prove the properties using a substantially smaller strengthening than the inductive engines. Moreover, PD-KIND is the only engine that can prove all properties that are already k-inductive. On the other hand, PD-KIND is also effective as a bug-finder due to the longer steps of k-induction. As an example of this, in one of the `hacms` examples, PD-KIND finds a counter-example of length 60 already at frame 15. The comparison is not exhaustive or definitive: we have not included tools such as KIND, and we used all the model checkers with default options. It is likely that they could be tuned to perform better on the particular benchmarks we have chosen.

VI. Conclusion

The k-induction principle is a popular method for proving safety of infinite-state systems. We have shown that k-induction can be more powerful than regular induction for expressive theories such as the theory of arrays. With this in mind, we proposed PD-KIND, a reformulation of the IC3 method that allows us to integrate k-induction into the method. We have implemented the new procedure in the SALLY tool, and the experimental evaluation shows that our prototype

[10] SALLY is open source and available at http://sri-csl.github.io/sally/.

[11] On the other hand, with no burden of proof-production, YICES2 is much faster on satisfiable queries.

[12] https://es-static.fbk.eu/people/griggio/vtsa2015/

[13] All benchmarks can be downloaded from http://csl.sri.com/~dejan/sally-benchmarks.tar.gz.

Fig. 4. Experimental evaluation. Each row corresponds to a different problem set. Each column corresponds to a different engine. Each table entry shows the number of problems that the engine solved, how many of those were valid and invalid, and the total time it took for the solved instances.

	Z3			SPACER			NUXMV			PD-KIND		
problem set	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)
approximate-agreement (9)	9	8/1	213	7	6/1	1150	9	8/1	2174	**9**	**8/1**	**164**
azadmanesh-kieckhafer (20)	20	17/3	3404	20	17/3	4678	20	17/3	294	**20**	**17/3**	**192**
cav12 (99)	69	48/21	2102	71	49/22	3529	**72**	**50/22**	**7443**	71	49/22	4990
conc (6)	4	4/0	128	4	4/0	655	**6**	**6/0**	**421**	4	4/0	270
ctigar (110)	64	44/20	1683	72	52/20	4249	76	56/20	1342	**77**	**57/20**	**2823**
hacms (5)	1	1/0	11	1	1/0	4	4	3/1	388	**5**	**3/2**	**1661**
lustre (790)	757	421/336	1888	763	427/336	2263	760	424/336	7660	**774**	**438/336**	**3494**
oral-messages (9)	9	7/2	16	9	7/2	44	9	7/2	161	**9**	**7/2**	**2**
tta-startup (3)	1	1/0	9	**1**	**1/0**	**8**	1	1/0	17	**1**	**1/0**	**8**
tte-synchro (6)	6	3/3	969	6	3/3	445	5	2/3	405	**6**	**3/3**	**21**
unified-approx (11)	8	5/3	2928	11	8/3	589	**11**	**8/3**	**139**	11	8/3	217

is effective at solving real-world problems. In addition, the new method is more powerful then k-induction, which is a novel and theoretically pleasing property: if the property being checked is k-inductive (for some k), then (modulo a requirement on the interpolation procedure) the method always terminates. It can also prove properties that are not k-inductive. When limiting the induction depth $k = 1$, the method can be seen as an effective instance of the IC3/PDR class of algorithms.

References

[1] M. H. Azadmanesh and R. M. Kieckhafer. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers*, 49(10):1031–1042, 2000.

[2] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, 1999.

[4] J. Birgmeier, A. R. Bradley, and G. Weissenbacher. Counterexample to induction-guided abstraction-refinement (CTIGAR). In *Computer Aided Verification*, pages 831–848, 2014.

[5] N. Bjørner, A. Gurfinkel, K. McMillan, and A. Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II*, pages 24–51. 2015.

[6] N. Bjørner and M. Janota. Playing with quantified satisfaction. Logic for Programming, Artificial Intelligence and Reasoning, 2015.

[7] A. R. Bradley. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, 2011.

[8] A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Verification, Model Checking, and Abstract Interpretation*, pages 427–442, 2006.

[9] A. Cimatti and A. Griggio. Software model checking via IC3. In *Computer Aided Verification*, pages 277–293, 2012.

[10] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 modulo theories via implicit predicate abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 46–61. 2014.

[11] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT5 SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–107. 2013.

[12] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient generation of Craig interpolants in satisfiability modulo theories. *ACM Transactions on Computational Logic*, 12(1):7, 2010.

[13] L. De Moura and N. Bjørner. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design*, pages 45–52, 2009.

[14] L. De Moura and D. Jovanović. A model-constructing satisfiability calculus. In *Verification, Model Checking, and Abstract Interpretation*, pages 1–12, 2013.

[15] L. De Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. *Lecture notes in computer science*, pages 14–26, 2003.

[16] A. F. Donaldson, L. Haller, D. Kroening, and P. Rümmer. Software verification using k-induction. In *Static Analysis*, pages 351–368. 2011.

[17] B. Dutertre. Yices 2.2. In *Computer Aided Verification*, pages 737–744, 2014.

[18] B. Dutertre. Solving exists/forall problems with Yices. In *SMT Workshop*, 2015.

[19] B. Dutertre, A. Easwaran, B. Hall, and W. Steiner. Model-based analysis of timed-triggered ethernet. In *Digital Avionics Systems Conference*, pages 9D2–1, 2012.

[20] N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *Formal Methods in Computer-Aided Design*, pages 125–134, 2011.

[21] S. Ghilardi and S. Ranise. MCMT: A model checker modulo theories. In *Automated Reasoning*, pages 22–29. 2010.

[22] K. Hoder and N. Bjørner. Generalized property directed reachability. In *Theory and Applications of Satisfiability Testing*, pages 157–171. 2012.

[23] A. Komuravelli, N. Bjørner, A. Gurfinkel, and K. L. McMillan. Compositional verification of procedural programs using horn clauses over integers and arrays. In *Formal Methods in Computer-Aided Design*, pages 89–96, 2015.

[24] A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-based model checking for recursive programs. In *Computer Aided Verification*, pages 17–34, 2014.

[25] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[26] N. A. Lynch. *Distributed algorithms*. 1996.

[27] J. McCarthy. Towards a mathematical science of computation. In *Program Verification*, pages 35–56. 1993.

[28] K. L. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1):101–121, 2005.

[29] P. Miner, A. Geser, L. Pike, and J. Maddalon. A unified fault-tolerance protocol. In *FORMATS/FTRTFT*, pages 167–182, 2004.

[30] A. M. Pitts. On an interpretation of second order quantification in first order intuitionistic propositional logic. *The Journal of Symbolic Logic*, 57(01):33–52, 1992.

[31] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Formal Methods in Computer-Aided Design*, pages 127–144, 2000.

[32] A. Stump, C. W. Barrett, and D. L. Dill. A decision procedure for an extensional theory of arrays. In *In 16th IEEE Symposium on Logic in Computer Science*, 2001.

[33] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1):3–27, 1988.