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, produced 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).

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 $s \models F$) if the formula evaluates to true under the state’s assignment.

A state-transition system is a pair $\mathcal{S} = (I, T)$, 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 $\mathcal{S}$ 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 $\sigma = \langle s_0, \ldots, s_k \rangle$ such that, $s = s_k$, the state $s_0$ satisfies $I$, and each $s_{i+1}$ is a successor of $s_i$. We call $\sigma$ 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 $s \models F$.

Given a state formula $P$ (the property), we want to determine whether all the reachable states of $\mathcal{S}$ satisfy $P$. If this is the case, $P$ is an invariant of $\mathcal{S}$, which we denote by $\mathcal{S} \models P$. We also write $\mathcal{S} \models_a^b P$ 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 $\neg P$.

1Some IC3 variants, such as PDR [22], are not guaranteed to terminate even if the property is already inductive.
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\(^2\) with respect to $\mathcal{S}$ if

\[
I(\bar{x}) \Rightarrow \mathcal{F}(\bar{x}),
\]

(init)

\[
\mathcal{F}(\bar{x}) \land T(\bar{x}, \bar{x}') \Rightarrow P(\bar{x}').
\]

(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 $\mathcal{S}$ if

\[
I(\bar{x}_0) \land \bigwedge_{i=0}^{l-1} T(\bar{x}_i, \bar{x}_{i+1}) \Rightarrow \mathcal{F}(\bar{x}_i), \text{ for } 0 \leq l < k, \text{ (k-init)}
\]

\[
\bigwedge_{i=0}^{k-1} (\mathcal{F}(\bar{x}_i) \land T(\bar{x}_i, \bar{x}_{i+1})) \Rightarrow P(\bar{x}_k), \text{ (k-cons)}
\]

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.\(^3\) 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.\(^4\)

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 $\mathcal{T}_{\text{arr}}$, and an extension of $\mathcal{T}_{\text{arr}}$ with array constants\([13]\), denoted by $\mathcal{T}^\ast_{\text{arr}}$. The theory $\mathcal{T}_{\text{arr}}$ 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 $\mathcal{T}^\ast_{\text{arr}}$ is obtained by adding a construct $c(v)$ that represents the constant array with value $v$, and the axiom $c(v)[i] = v$.

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 $\mathcal{T}^\ast_{\text{arr}}$ as arrays that map integers to Booleans, and define set operations as $x \in a \equiv a[x]$, $\emptyset \equiv c(\text{false})$, $a \cup \{x\} \equiv \text{WRITE}(a, x, \text{true})$.

\(^2\)This is the same idea as induction relative to $\mathcal{F}$ used in [7].

\(^3\)If $P$ is $k$-inductive then $P \land a \circ P \land \ldots a \circ P \ldots a \circ P$ is inductive, where $\circ$ stands for “next state”.

\(^4\)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.

---

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.\(^5\) 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 $\mathcal{T}_{\text{arr}}$ 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 $P \land \bigwedge_{i=1}^{N-1} (a[k] = 0 \lor i < k)$.

Lemma III.1. There exists a sequence of state-transition systems $\mathcal{S}_N$ and a property $P$, such that for any $N$ the property $P$ is $N$-inductive in $\mathcal{S}_N$, 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 $a[0], \ldots, a[\ell], \ldots$ 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 $\mathcal{T}^\ast_{\text{arr}}$ with initial states defined

\(^5\)For example, $\exists i . a[i] = 0$ does not have a quantifier-free equivalent.
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 $T^*_{arr}$.

Lemma III.2. There exists a state-transition system $S$ 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

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

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 $F \equiv \text{true}$, we omit it and simply write $T^k$.

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

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

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) $A(\vec{x}) \land T[B]^k(\vec{x}, \vec{w}, \vec{y}) \Rightarrow J(\vec{y})$, 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) $G(\vec{x}) \Rightarrow \exists \vec{w}, \vec{y} . T[B]^k(\vec{x}, \vec{w}, \vec{y}) \land C(\vec{y})$.

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 $(A \land T[B]^k)[\vec{w}/v(\vec{w}), \vec{y}/v(\vec{y})]$ 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 $A \land T[B]^k$ and precise generalizations by eliminating $\vec{w}$ and
\( \bar{y} \) from \( T[B]^k \land B \). 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. \(^6\)

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 \( \mathcal{S} = \{ I, T \} \) defined as \( I \equiv (x = 0), T \equiv (x' = x + 1) \) where \( x \) is a real-valued variable. Let \( P \) be the formula \( 0 \leq x \land x \geq 1 \). To check whether \( P \) is inductive, we can ask the following satisfiability query to the SMT solver

\[
\begin{align*}
A_1 & \equiv \neg & (0 \leq x \land x \geq 1) & \land & (x' = x + 1) & \land & \neg (0 \leq x' \land x' \geq 1) .
\end{align*}
\]

This formula is satisfiable in a model \( x \mapsto -0.5, x' \mapsto 0.5 \). We can then check whether \( G \) intersects with the initial states and whether \( G \) is reachable in one step, by making two separate queries

\[
\begin{align*}
A_2 & \equiv \neg & (x = 0) & \land & (x' = x) & \land & (0 < x' < 0) , \\
B_2 & \equiv \neg & (x = 0) & \land & (x' = x + 1) & \land & (0 < x' < 0) .
\end{align*}
\]

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

**B. Reachability**

**Problem IV.1 (k-reaching).** Given a state formula \( F \) that is not reachable in fewer than \( k \) 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 \( \mathcal{S} \) and state formula \( F \) that is unreachable in \( \leq k \) steps (system is \( k \)-inconsistent with \( F \)), a state formula \( J \) is a state \( k \)-interpolant for \( F \) if

\[
\mathcal{S} \models^k J , \quad \text{\( J \) and \( F \) are inconsistent}.
\]

\(^6\)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 \( \bar{S} \) and \( F \) are \( k \)-inconsistent, then \( I(\bar{x}) \land T^k(\bar{x}, \bar{w}, \bar{x}') \land F(\bar{x}') \) is unsatisfiable for \( 0 \leq i \leq k \). From these inconsistencies we can obtain interpolants \( J_0, \ldots, J_k \), and the formula \( J \equiv (J_0 \lor \ldots \lor J_k) \) 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 \( \mathcal{S} \models^i \mathcal{R}_i \). Unlike IC3/PDR, we do not require the frames to be monotonic: we may have \( \mathcal{R}_{i+1} \subseteq \mathcal{R}_i \).

This setup allows us to build \( k \)-interpolants efficiently provided an extra local condition holds. If \( k = 0 \), we just take the interpolant of \( I \land T^0 \land F \). 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 \( I \land T^0 \land F \) and \( \mathcal{R}_{k-1} \land T \land F \) are inconsistent. We can then obtain interpolants \( J_1 \) and \( J_2 \) for these two formulas and \( (J_1 \lor J_2) \) is a \( k \)-interpolant for \( F \). This \( k \)-interpolant, which we denote by EXPLAIN\( (\mathcal{S}, k, F) \), 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.

**Algorithm 1 Check \( k \)-reachability of \( F \).**

*Require: \( \mathcal{S} \models^i \mathcal{R}_i \) for \( 0 \leq i \leq k \), \( \mathcal{S} \models^k \neg F \).*

*Ensure: \( \mathcal{S} \models^i \mathcal{R}_i \) for \( 0 \leq i \leq k \). If not reachable, \( \mathcal{R}_{k-1} \land T \land F \) is unsatisfiable.*

1. function REACHABLE(\( \mathcal{S}, k, F \))
2. if \( k = 0 \) then return CHECK-SAT\( (I, T^0, F) \)
3. loop
4. if CHECK-SAT\( (\mathcal{R}_{k-1}, T, F) \) then
5. \( G \leftarrow \text{GENERALIZE}(\mathcal{R}_{k-1}, T, F) \)
6. if REACHABLE\( (\mathcal{S}, k - 1, G) \) then
7. return true
8. else
9. \( E \leftarrow \text{EXPLAIN}(\mathcal{S}, k - 1, G) \)
10. \( \mathcal{R}_{k-1} \leftarrow \mathcal{R}_{k-1} \cup \{ E \} \)
11. else return false

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

\(^7\)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 $R_{k-1}$. If no such transition is possible, then $F$ is not reachable. Otherwise, we get a state $s$ that satisfies $R_{k-1}$ 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 $R_{k-1}$. 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 $(r, l) = \text{REACHABLE}(\mathcal{G}, k_1, k_2, F)$. 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 $\mathcal{G} = (I, T)$. 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 $\mathcal{G}$ 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 $1 \leq k \leq n+1$. 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 counterexample 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 $\mathcal{F}_{\text{ABS}}$-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 $\mathcal{G}$ 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 $\mathcal{F}_{\text{ABS}} \in \mathcal{F}_{\text{ABS}}$ information about a potential counter-example to $P$ that the formula $\mathcal{F}_{\text{ABS}}$ eliminates. The set $\mathcal{F}_{\text{ABS}}$ and this additional information is represented in the form of an induction frame. Let $\mathcal{F}$ denote the set of all state formulas in our theory.

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

1) $\mathcal{F}_{\text{ABS}}$ is valid up to $n$ steps and refutes $\mathcal{F}_{\text{CEX}}$, and
2) $\mathcal{F}_{\text{CEX}}$-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., $\mathcal{F}_{\text{ABS}} = \{ \mathcal{F}_{\text{ABS}} \mid (\mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{CEX}}) \in \mathcal{F} \}$. 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 $1 \leq k \leq n + 1$. 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 $(\mathcal{F}_{\text{ABS}}, \mathcal{F}_{\text{CEX}})$ from $\mathcal{F}$. Both $\mathcal{F}_{\text{ABS}}$ and $\mathcal{F}_{\text{CEX}}$ hold up to index $n$ in $\mathcal{G}$.

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

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

---

**Algorithm 2 Main PD-KIND procedure.**

**Require:** $\mathcal{G} = (I, T)$ and $I \Rightarrow P$

1. **function** PD-KIND($\mathcal{G}, P$)
2. \hspace*{0.5cm} $n \leftarrow 0$
3. \hspace*{0.5cm} $\mathcal{F} \leftarrow \{(P, \neg P)\}$
4. **loop**
5. \hspace*{1cm} pick $k$-induction depth $1 \leq k \leq n + 1$
6. \hspace*{1cm} $(\mathcal{F}, G, n_p) \leftarrow \text{PUSH}(\mathcal{G}, \mathcal{F}, P, n, k)$
7. \hspace*{1cm} if $P$ marked invalid then return invalid
8. \hspace*{1cm} if $\mathcal{F} = G$ then return valid
9. \hspace*{1cm} $n \leftarrow n_p$
10. \hspace*{0.5cm} $\mathcal{F} \leftarrow G$
states of $\mathcal{G}$. Because we know that $F_{\text{CEX}}$ is not $n$-reachable, $G_{\text{CEX}}$ can’t be reached in less than $n - k + 1$ steps. So we check reachability of $G_{\text{CEX}}$ at positions $n - k + 1 \ldots n$. If $G_{\text{CEX}}$ is reachable, then so is $\neg P$ and we mark $P$ as invalid. Otherwise, we call the EXPLAIN procedure, which returns a new fact $G_{\text{ABS}}$ that eliminates $G_{\text{CEX}}$. The new fact $G_{\text{ABS}}$ is true up to position $n$, and refutes $G_{\text{CEX}}$, so we can add the new induction obligation $(G_{\text{ABS}}, G_{\text{CEX}})$ to $F$, strengthening $F$, and try again with a potential counter-example eliminated.

In the remaining case, we have a counterexample $m_{\text{CTI}}$ to the $k$-inductiveness of $F_{\text{ABS}}$. Since the query at line 15 is not satisfied and $F_{\text{ABS}} \Rightarrow \neg F_{\text{CEX}}$, we know that $\neg F_{\text{CEX}}$ is $F_{\text{ABS}}$-inductive. We first apply generalization to $m_{\text{CTI}}$ to construct a formula $G_{\text{CTI}}$. From any state that satisfies $G_{\text{CTI}}$, we can reach $\neg F_{\text{ABS}}$ in $k$ steps. If $G_{\text{CTI}}$ is reachable in $\mathcal{G}$ then $\neg F_{\text{ABS}}$ is also reachable, so $F_{\text{ABS}}$ 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 $G_{\text{CTI}}$ at positions $n - k + 1, \ldots, n$. If $G_{\text{CTI}}$ is reachable, we can’t push $F_{\text{ABS}}$. 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 $G_{\text{CTI}}$ is not reachable then we strengthen $F_{\text{ABS}}$ with a new fact $G_{\text{ABS}}$ 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 $\mathcal{G}$ and that this requires at least $n$ 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 $F$ of $k$-induction assumptions for $\mathcal{G}$, and the shortest refutation length $n_p$ for any $F_{\text{ABS}} \in F_{\text{ABS}}$ that was not successfully pushed. The procedure does not only add to the original set $F$, 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 $F_{\text{ABS}}$ is valid up to position $n_p - 1 \geq n$, and the facts in $G_{\text{ABS}}$ are valid up to position $n_p \geq n + 1$.

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.

---

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

---

**Algorithm 3** Push $F$ with $k$-induction.

**Require:** $F$ is a valid frame for $P$ at position $n$, $1 \leq k \leq n + 1$, $(P, \neg P) \notin F$.

**Ensure:** $F$ is a valid frame for $P$ at position $n_p - 1 \geq n$, $\mathcal{G} \subseteq F$ is $F_{\text{ABS}}$-inductive, and $P$ marked invalid or $(P', \neg P', 0) \notin F_p$.

```
function PUSH($\mathcal{G}, F, P, n, k$)
  push elements of $F$ to $Q$. ▷ $Q$ is a priority queue.
  $\mathcal{G} \leftarrow \emptyset$. ▷ Pushed facts, i.e., $G_{\text{ABS}}$ is $F_{\text{ABS}}$-inductive.
  while $P$ not marked invalid, $Q$ not empty do
    pop ($F_{\text{ABS}}, F_{\text{CEX}}$) from $Q$
    if sat$_{\text{CTI}}(m_{\text{CTI}})$ ▷ Is $F_{\text{ABS}}$ $F_{\text{ABS}}$-inductive?
      (sat$_{\text{CTI}}, m_{\text{CTI}}$) $\leftarrow$ CHECK-SAT($F_{\text{ABS}}, T[F_{\text{ABS}}]^k, G_{\text{CTI}}$)
      if not sat$_{\text{CTI}}$ then
        $\mathcal{G} \leftarrow \mathcal{G} \cup \{(F_{\text{ABS}}, F_{\text{CEX}})\}$. ▷ $G_{\text{ABS}}$ is $F_{\text{ABS}}$-inductive.
        continue
    (sat$_{\text{CTI}}, m_{\text{CTI}}$) $\leftarrow$ CHECK-SAT($F_{\text{ABS}}, T[F_{\text{ABS}}]^k, F_{\text{CEX}}$)
    if sat$_{\text{CTI}}$ then
      $G_{\text{CTI}}$ $\leftarrow$ GENERALIZE($m_{\text{CTI}}, T^k, F_{\text{CEX}}$)
      if REACHABLE($\mathcal{G}, n - k + 1, n, G_{\text{CTI}}$) then
        mark $P$ invalid ▷ $I \Rightarrow G_{\text{CTI}} \Rightarrow k F_{\text{CEX}} \Rightarrow P$.
      else
        $G_{\text{ABS}}$ $\leftarrow$ EXPLAIN($\mathcal{G}, n, F_{\text{CEX}}$)
        $F \leftarrow F \cup \{(G_{\text{ABS}}, F_{\text{CEX}})\}$. ▷ Eliminate CEX.
        push ($G_{\text{ABS}}, G_{\text{CTI}}$), ($F_{\text{ABS}}, F_{\text{CEX}}$) to $Q$
        continue
    $G_{\text{CTI}}$ $\leftarrow$ GENERALIZE($m_{\text{CTI}}, T^k, \neg F_{\text{ABS}}$)
    (sat$_{\text{CTI}}, m_{\text{CTI}}$) $\leftarrow$ REACHABLE($\mathcal{G}, n - k + 1, n, G_{\text{CTI}}$)
    if sat$_{\text{CTI}}$ then
      $I \Rightarrow G_{\text{CTI}} \Rightarrow k F_{\text{CEX}} \Rightarrow P$.
      (sat$_{\text{CTI}}, m_{\text{CTI}}$) $\leftarrow$ REACHABLE($\mathcal{G}, n + 1, n, G_{\text{CTI}}$)
      if $n_p \leq \min(n_p, n_{\text{CTI}})$
        $F \leftarrow F \cup \{(\neg F_{\text{CEX}}, F_{\text{CEX}})\}$
        $\mathcal{G} \leftarrow \mathcal{G} \cup \{(\neg F_{\text{CEX}}, F_{\text{CEX}})\}$
    else
      $I \not\Rightarrow G_{\text{CTI}} \Rightarrow k F_{\text{CEX}}.
      G_{\text{ABS}}$ $\leftarrow$ EXPLAIN($\mathcal{G}, n, G_{\text{CTI}}$) ▷ $G_{\text{ABS}}$ is $G_{\text{CTI}}$.
      $G_{\text{ABS}}$ $\leftarrow$ $F_{\text{ABS}} \land G_{\text{ABS}}$. ▷ $G_{\text{ABS}}$ is $\neg F_{\text{CEX}}$.
      $F \leftarrow F \cup \{(G_{\text{ABS}}, F_{\text{CEX}})\} \setminus \{(F_{\text{ABS}}, F_{\text{CEX}})\}$
      push ($G_{\text{ABS}}, F_{\text{CEX}}$) to $Q$.

return ($F, G, n_p$).
```
to the property. On the other hand, if PD-KIND terminates when the inductive frames become equal, i.e. $F = G$, then $F\text{abs}$ 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 can be added to the frame 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 $m_{\text{ex}}$ 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 then, 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 $F = G$.

**V. EXPERIMENTAL EVALUATION**

We have implemented PD-KIND in the SALLY model-checker. 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. 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 benchmark results are related to fault-tolerant algorithms (oral-messages [25], tte-synchr0 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. 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.

To put the results in context, we compare PD-KIND with other state-of-the-art, infinite-state model checkers, namely, Z3 [22], NU/XMV [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. For reference, we include additional experimental data in the appendix.

**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.

---

9SALLY is open source and available at http://sri-csl.github.io/sally/.

10On the other hand, with no burden of proof-production, YICES2 is much faster on satisfiable queries.


12All benchmarks can be downloaded from http://csl.sri.com/~dejan/sally-benchmarks.tar.gz.
and the experimental evaluation shows that our prototype is effective at solving real-world problems. In addition, the new method is more powerful than 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


APPENDIX

A. Additional Experimental Results

Table 5 reports the results with addition of the PD-KIND method where the \( k\)-induction depth is restricted to 1, i.e. we always pick \( k = 1 \). This variant, that we refer to as PD-KIND\(_1\), can be seen as an instance of the IC3/PDR class of algorithms in the same codebase.

Figure 6 presents a more detailed comparison of PD-KIND with the other engines. Figure 7 compares basic \( k\)-induction with other engines on the same set of benchmarks.

B. Relative Power of Induction and \( k\)-Induction: Proofs

Lemma III.1. There exists a sequence of state-transition systems \( \Sigma_N \) and a property \( P \), such that for any \( N \) the property \( P \) is \( N \)-inductive in \( \Sigma_N \), but the shortest inductive strengthening of \( P \) has a size larger than \( N \).

Proof. We rely on Example III.1 to prove the statement. Let \( Q \) be the smallest inductive invariant of the system \( \Sigma_{\exp} \) that implies \( P \), meaning it is in the theory \( \mathcal{T}_{\text{arr}} \), quantifier-free, and only contains variables \( a, i, j \). In order to be inductive, the formula \( Q \) must allow all states reachable in \( \Sigma_{\exp} \) and not allow any states that could transition into a state that falsifies \( P \).

We show that such a formula \( Q \) must contain at least \( N \) index terms (terms that are used to index an array, including \( i, j \), and any constants). Assume the opposite, i.e., that \( Q \) contains less than \( N \) index terms, and consider a state \( s_1 \) that interprets \( i \) as \( N - 1 \) and \( a[0], \ldots, a[N - 1] \) as 0. Since \( Q \) contains less than \( N \) index terms, there is at least one value \( k \in \{0, 1, \ldots, N - 1\} \) that is not mapped to any index term of \( Q \) in state \( s_1 \). We can construct a state \( s_2 \) equivalent to \( s_1 \) everywhere except for the interpretation of \( a[k] \), where it assigns 1. It is easy to show by induction on the structure of \( Q \) that the interpretation of \( Q \) doesn’t change, i.e., that \( s_2 \models Q \). But, from \( s_2 \), the system can transition into a state where \( a[0] = 1 \), falsifying \( P \). This shows that \( Q \) cannot be the invariant we are looking for, and by contradiction, it follows that \( Q \) must have at least \( N \) index terms.

The size of the \( k \)-inductive invariant \( P \) is constant while, from above, the size of \( Q \) must be at least \( N \), thus proving the statement.

Lemma III.2. There exists a state-transition system \( \Sigma \) and a property \( P \), such that the property \( P \) is \( k \)-inductive for \( k > 1 \) but there is no inductive strengthening of \( P \).

Proof. We rely on Example III.2 to prove the statement. Let \( Q \) be an inductive invariant of system \( \Sigma_{\pow} \) that implies \( P \). Formula \( Q \) is in theory \( \mathcal{T}_{\text{arr}} \), quantifier-free, and only contains variables \( a, J, i, j \). In order to be inductive, \( Q \) must allow all models reachable in \( \Sigma_{\exp} \) and not allow any models that could transition into a state that falsifies \( P \).

Let’s assume that such a formula \( Q \) exists. We will construct states \( s_1, s_2, \) and \( s_3 \) such that

- \( s_1 \) and \( s_2 \) are reachable in \( \Sigma_{\pow} \) and therefore satisfy any \( Q \);
- \( Q \) can not distinguish between \( s_2 \) and \( s_3 \);
- \( s_3 \) has a successor state that falsifies \( P \).

This implies that there is no inductive strengthening of \( P \).

Consider a state \( s_1 \) that assigns \( c \) to 1, \( i \) and \( j \) to 0, \( a_{\text{exp}} \) to \( a[0] \), and \( J \) to \{0\}, as sketched in Figure 8. This model is reachable in \( \Sigma_{\pow} \) (e.g., from initial states, by writing to \( a[0] \)). Let \( V \) be the set containing the interpretation (in \( s_1 \)) of all the constants and index terms of \( Q \). The set \( V \) is finite, so we can pick an integer \( k \) larger than any element of \( V \). We now define the states \( s_2 \) and \( s_3 \) to be the same as \( s_1 \), except as follows.

\[ s_2(a)[k] = 0, \quad s_2(J) = \{0, k\}, \quad s_3(J) = \{0, k\}. \]

The state \( s_2 \) is also a reachable state of \( \Sigma_{\pow} \) (e.g., from initial states, by writing to \( a[k] \), then to \( a[0] \) again). This means that \( s_2 \) satisfies \( Q \). The state \( s_3 \), on the other hand, can transition into a state where the property does not hold and should therefore not satisfy \( Q \). But, we will show that \( Q \) cannot distinguish states \( s_2 \) and \( s_3 \). We do so by proving the following by induction on the structure of \( Q \):

1) All integer terms, Boolean terms, and Boolean arrays evaluate to the same value in \( s_2 \) and \( s_3 \).
2) All integer-array terms \( t \) obtained from \( a \) (using zero or more WRITE applications) evaluate to the same value in \( s_2 \) and \( s_3 \), except at index \( k \) where \( s_2(b)[k] = 0 \) and \( s_3(b)[k] = k \).
3) All integer-array terms not obtained from \( a \), i.e. all arrays obtained from constant arrays, evaluate to the same value in \( s_2 \) and \( s_3 \).

The induction is a routing exercise that considers all term constructors available, and is based on the intuition that terms in \( Q \) can only “see” the array elements at indices in \( V \) and the index \( k \) is beyond reach. The more delicate cases are array equalities of the form \( t_1 = t_2 \). If both \( t_1 \) and \( t_2 \) are Boolean arrays, it follows from (1) above that \( t_1 \) and \( t_2 \) have the same interpretation in \( s_1 \) and \( s_2 \), and hence the same holds for the equality too. If \( t_1 \) and \( t_2 \) are integer arrays, then we distinguish the following cases:

1) If \( t_1 \) and \( t_2 \) are both obtained from constant arrays then, by (3) above, they have the same interpretation in \( s_1 \) and \( s_2 \), and hence the same holds for the equality too.
2) If \( t_1 \) and \( t_2 \) are both obtained from \( a \), by (2) above the equality can not depend on index \( k \) and therefore the equality has the same interpretation in \( s_1 \) and \( s_2 \).
3) If \( t_1 \) is obtained from \( a \) and \( t_2 \) is obtained from constant arrays, then the equality must evaluate to false in both \( s_1 \) and \( s_2 \). This is because \( t_1 \) contains an infinite number of values, while \( t_2 \) can only contains a finite number of values.

The state \( s_3 \) can transition into a state that falsifies \( P \), e.g., by picking \( i = 0 \), \( j = k \in J \), and then writing \( k \) into \( a[0] \). This means that \( Q \) can not be the invariant we are looking for and thus proves the statement.
Fig. 5. 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.

<table>
<thead>
<tr>
<th>problem set</th>
<th>S3</th>
<th>SPACER</th>
<th>NU XMV</th>
<th>PD-KIND∞</th>
<th>PD-KIND1</th>
</tr>
</thead>
<tbody>
<tr>
<td>approximate-agreement (9)</td>
<td>9</td>
<td>8/1 213</td>
<td>7 6/1 1150</td>
<td>9</td>
<td>8/1 2174</td>
</tr>
<tr>
<td>azadmanesh-kieckhafer (20)</td>
<td>20</td>
<td>17/3 3404</td>
<td>20 17/3 4678</td>
<td>20</td>
<td>17/3 294</td>
</tr>
<tr>
<td>cav12 (99)</td>
<td>69</td>
<td>48/21 2102</td>
<td>71 49/22 3529</td>
<td>72</td>
<td>50/22 7443</td>
</tr>
<tr>
<td>conc (6)</td>
<td>4</td>
<td>4/0 128</td>
<td>4 4/0 655</td>
<td>6</td>
<td>6/0 421</td>
</tr>
<tr>
<td>cigar (110)</td>
<td>64</td>
<td>44/20 1683</td>
<td>72 52/20 4249</td>
<td>76</td>
<td>56/20 1342</td>
</tr>
<tr>
<td>hacms (5)</td>
<td>1</td>
<td>1/0 11</td>
<td>1 1/0 4</td>
<td>4 3/1 388</td>
<td>5</td>
</tr>
<tr>
<td>oral-messages (9)</td>
<td>9</td>
<td>7/2 16</td>
<td>9 7/2 44</td>
<td>9 7/2 161</td>
<td>9</td>
</tr>
<tr>
<td>tta-startup (3)</td>
<td>1</td>
<td>1/0 9</td>
<td>1 1/0 8</td>
<td>1 1/0 17</td>
<td>1 1/0 8</td>
</tr>
<tr>
<td>tte-synchro (6)</td>
<td>6</td>
<td>3/3 969</td>
<td>6 3/3 445</td>
<td>5 2/3 405</td>
<td>6</td>
</tr>
<tr>
<td>unified-approx (11)</td>
<td>8</td>
<td>5/3 2928</td>
<td>11 8/3 589</td>
<td>11</td>
<td>8/3 139</td>
</tr>
</tbody>
</table>

C. k-reachability: Proofs

Lemma IV.1. Starting from a fixed finite frame sequence \( R_0, R_1, \ldots \), 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.

Proof. Let us assume that the interpolation is finite-covering. If the only way to add new formulas to the reachability frames is through the \EXPLAIN\ procedure then, we can show by induction on \( k \) that the number of possible \( R_k \) sets is finite. This holds for \( k = 0 \), since we never add formulas to \( R_0 \). For \( k > 0 \), since there are finitely many possible \( R_{k-1} \) sets, and \EXPLAIN\ interpolates from \( I \) and \( R_{k-1} \), then there are finitely many possible \( k \)-interpolants that we add to \( R_k \), showing the case. □

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 \( R_{k-1} \). In addition, if the interpolation or the generalization procedure is finite-covering, then the procedure always terminates.

Proof. There are two parts of the statement, correctness and termination, each of which can be proved by induction. Correctness follows directly by following the annotations in the procedure description. For termination we consider the two cases. If the generalization procedure is finite-covering, then there is only a finite number of previous step generalizations \( G \) that we need to eliminate by recursive calls. Each of those calls terminate and therefore the procedure itself terminates. If the \( k \)-interpolation procedure behind \EXPLAIN\ is finite-covering then, similarly, we can only learn a finite number of explanations, which again implies termination. □
Fig. 6. Scatterplot comparison of PD-KIND∞ with Z3, SPACER, NUXMV, and PD-KIND₁. Each point represents the time the two solver took to solve a problem. Red points represent invalid benchmarks, green points represent valid benchmarks.
Fig. 7. Scatterplot comparison of $k$-induction with PD-KIND$_{\infty}$, PD-KIND$_1$, Z3, SPACER, and NUXMV. Each point represents the time the two solver took to solve a problem. Red points represent invalid benchmarks, green points represent valid valid benchmarks.
Fig. 8. States $s_1, s_2, s_3$

<table>
<thead>
<tr>
<th>States</th>
<th>Interpretation of $\alpha$</th>
<th>Interpretation of $J$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$\ldots -3 -2 -1 0 1 2 3 \ldots k \ldots$</td>
<td>$\ldots f f t f f \ldots f \ldots {0}$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$\ldots -3 -2 -1 0 1 2 3 \ldots 0 \ldots$</td>
<td>$\ldots f f t f f \ldots t \ldots {0,k}$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$\ldots -3 -2 -1 0 1 2 3 \ldots k \ldots$</td>
<td>$\ldots f f t f f \ldots t \ldots {0,k}$</td>
</tr>
</tbody>
</table>