PDR for k-safety

No Author Given

No Institute Given

Abstract. We introduce an algorithm for k-safety. Based on PDR.

1 Introduction

2 Preliminaries

In this section, we present notations and background that is required for the description of our algorithm.

Safety verification. A transition system T is a tuple $(X, \mathit{Init}, \mathit{Tr}, \mathit{Bad})$, where X is a set of variables that defines the states of the system (i.e., 2^X), Init and Bad are formulas with variables in X denoting the set of initial states and bad states, respectively, and Tr is a formula with free variables in $X \cup X'$, denoting the transition relation. A state $s \in 2^X$ is said to be reachable in T if and only if (iff) there exists a state $s_0 \in \mathit{Init}$, and $(s_i, s_{i+1}) \in \mathit{Tr}$ for $0 \leq i \leq N$, and $s = s_N$. For simplicity of presentation, we assume that the initial states do not include bad states, that is $\mathit{Init} \Rightarrow \neg \mathit{Bad}$.

An *inductive invariant* is a formula Inv that satisfied:

$$\mathit{Init}(X) \Rightarrow Inv(X) \qquad\qquad \mathit{Inv}(X) \wedge \mathit{Tr}(X, X') \Rightarrow \mathit{Inv}(X') \tag{1}$$

A transition system T is SAFE iff there exists an inductive invariant Inv s.t. $Inv(X) \Rightarrow \neg Bad(X)$. In this case we say that Inv as a *safe* inductive invariant. A formula Inv is said to be inductive *relative* to a formula F if Inv satisfies (1) with $\mathit{Tr}(X, X')$ replaced by $F(X) \wedge \mathit{Tr}(X, X')$.

A transition system T is UNSAFE iff there exists a state $s \in \mathit{Bad}$ s.t. s is reachable. Equivalently, T is UNSAFE iff there exists a number N such that the following formula is satisfiable:

$$\mathit{Init}(X_0) \wedge \left(\bigwedge_{i=0}^{N-1} \mathit{Tr}(X_i, X_{i+1}) \right) \wedge \mathit{Bad}(X_N) \tag{2}$$

where $X_i = \{a_i \mid a \in X\}$ is a copy of the variables used to represent the state of the system after the execution of i steps. We use $\textsc{Bmc}(T, k)$ to denote a procedure that checks satisfiability of Formula 2. When T is UNSAFE and $s_N \in \mathit{Bad}$ is the reachable state, the path from $s_0 \in \mathit{Init}$ to s_N is called a *counterexample* (CEX).

The *safety* verification problem is to decide whether a transition system T is SAFE or UNSAFE, i.e., whether there exists a safe inductive invariant or a counterexample.

Input: A safety problem $\langle Init(X), Tr(X, X'), Bad(X)\rangle$.
Output: *Unreachable* or *Reachable*
Data: A cex queue $\mathcal{Q}$, where $c \in \mathcal{Q}$ is a pair $\langle m, i\rangle$, m is a cube over state variables, and $i \in \mathbb{N}$. A level N. A trace $F_0, F_1, \ldots$
Initially: $\mathcal{Q} = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$.
repeat

- **Unreachable** If there is an $i < N$ s.t. $F_{i+1} \subseteq F_i$ **return** *Unreachable*.
- **Reachable** If there is an m s.t. $\langle m, 0\rangle \in \mathcal{Q}$ **return** *Reachable*.
- **Unfold** If $F_N \to \neg Bad$, then set $N \leftarrow N + 1$.
- **Candidate** If for some m, $m \to F_N \wedge Bad$, then add $\langle m, N\rangle$ to $\mathcal{Q}$.
- **Predecessor** If $\langle m, i+1\rangle \in \mathcal{Q}$ and there are m_0 and m_1 s.t. $m_1 \to m$, $m_0 \wedge m_1'$ is satisfiable, and $m_0 \wedge m_1' \to F_i \wedge Tr \wedge m'$, then add $\langle m_0, i\rangle$ to $\mathcal{Q}$.
- **NewLemma** For $0 \leq i < N$: given a candidate model $\langle m, i+1\rangle \in \mathcal{Q}$ and clause φ, such that $\varphi \to \neg m$, if $Init \to \varphi$, and $\varphi \wedge F_i \wedge Tr \to \varphi'$, then add φ to F_j, for $j \leq i+1$.
- **ReQueue** If $\langle m, i\rangle \in \mathcal{Q}$, $0 < i < N$ and $F_{i-1} \wedge Tr \wedge m'$ is unsatisfiable, then add $\langle m, i+1\rangle$ to $\mathcal{Q}$.
- **Push** For $0 \leq i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $Init \to \varphi$ and $\varphi \wedge F_i \wedge Tr \to \varphi'$, then add φ to F_j, for each $j \leq i+1$.

until ∞;

Algorithm 1: IC3/PDR.

An *inductive trace*, or simply a trace, is a sequence of formulas $[F_0, \ldots, F_k]$ that satisfy:

$$Init \Rightarrow F_0 \qquad \forall 0 \leq i < k \cdot F_i(X) \wedge Tr(X, X') \to F_{i+1}(X') \tag{3}$$

An inductive trace is *monotone* if $\forall 0 \leq i < N \cdot F_i \Rightarrow F_{i+1}$ and safe when $\forall i \cdot F_i \Rightarrow \neg Bad$.

Note that for a monotone inductive trace $[F_0, \ldots, F_k]$, F_i is a bounded invariant w.r.t. i for all $0 \leq i \leq k$.

3 IC3 and PDR

The finite state model checking algorithm IC3 was introduced in [**?**] and its variant PDR in [**?**]. It maintains sets of clauses $F_0, \ldots, F_i, \ldots, F_N$, called a *trace*, that are properties of states reachable in i steps from the initial states $Init$. Elements of F_i are called *lemmas*. In the following, we assume that F_0 is initialized to $Init$. After establishing that $Init \to \neg Bad$, the algorithm maintains the following invariants (for $0 \leq i < N$):

Invariant 1

$$F_i \to \neg Bad \qquad F_i \to F_{i+1} \qquad F_i \wedge Tr \to F'_{i+1}$$

That is, each F_i is safe, the trace is monotone, and F_{i+1} is inductive relative to F_i. In practice, the algorithm enforces monotonicity by maintaining $F_{i+1} \subseteq F_i$.

Alg. 1 summarizes, in a simplified form, a variant of the IC3 algorithm. The algorithm maintains a queue of counter-examples Q. Each element of Q is a tuple $\langle m, i \rangle$ where m is a monomial over $\boldsymbol{v}$ and $0 \leq i \leq N$. Intuitively, $\langle m, i \rangle$ means that a state m can reach a state in Bad in $N - i$ steps. Initially, Q is empty, $N = 0$ and $F_0 = Init$. Then, the rules are applied (possibly in a non-deterministic order) until either **Unreachable** or **Reachable** rule is applicable. **Unfold** rules extends the current trace and increases the level at which counterexample is searched. **Candidate** picks a set of bad states. **Predecessor** extends a counter-example from the queue by one step. **NewLemma** blocks a counterexample and adds a new lemma. **ReQueue** moves the counterexample to the next level. Finally, **Push** generalizes a lemma inductively. A typical schedule of the rules is to first apply all applicable rules except for **Push** and **Unfold**, followed by **Push** at all levels, then **Unfold**, and then repeating the cycle.

Queue. The queue is ordered by the level:

$$\langle m, i \rangle < \langle n, j \rangle \iff i < j \tag{4}$$

This drives the algorithm to the shortest counterexample.

Inductive Generalization. The **NewLemma** and **Push** rules are based on the principle of inductive generalization. Let $F_0, \ldots, F_i, \ldots, F_N$ be a valid trace, and let φ be a clause that is *relatively inductive* to F_i:

$$Init \implies \varphi \qquad\qquad \varphi \wedge F_i \wedge Tr \implies \varphi' \tag{5}$$

Let $\boldsymbol{G} = G_0, \ldots, G_N$ be defined as follows:

$$G_j = \begin{cases} F_j \cup \{\varphi\} & \text{if } j \leq i+1 \\ F_j & \text{if } i+1 < j \leq N \end{cases} \tag{6}$$

Then $\boldsymbol{G}$ is a valid trace. The proof is by induction on i and follows from monotonicity of the trace.

Generalizing predecessors. The **Predecessor** rule picks a predecessor m_0 in Tr of some (partial) state m. While it is possible to simply pick a predecessor state, the rule attempts to find a generalized predecessor instead. The conditions of the rule is sufficient to ensure that m_0 is an implicant of $\psi = (F_i \wedge \exists X' \cdot (Tr \wedge m'))$. Finding a prime implicant of ψ would have been even better, but is too expensive in practice.

Propagating lemmas. The **Push** rule propagates lemmas to higher level, optionally generalizing them as possible. This makes the trace "more" inductive, eventually leading to convergence.

Long counterexamples. The **ReQueue** rule lifts blocked counterexamples to higher levels. As a side-effect, it makes it possible to discover counterexamples longer than the current exploration bound N. For example, assume that m is blocked at level i. This means that there is a path of length $N - i$ from m to Bad (but no path of length at most i from $Init$ to m). Assume that **ReQueue** lifted m to level $j > i$, and then m was reachable from $Init$. Then, the discovered counterexample is a concatenation of a path of length k from $Init$ to m and a path of length $N - i$ from m to Bad. The total length of the counterexample is $(N - i + k)$ which is bigger than N.

4 *k*IC3

We present a variant of PDR which is designed for k-safety verification. For simplicity, we present the algorithm for $k = 2$. However, the extension to any k is natural.

Our goal is information flow analysis, we therefore first go over the definition of self-composition.

When using self-composition, information flow is tracked over an execution of two copies of the transition system, T and T_d. Let us denote $X_d := \{x_d \mid x \in X\}$ as the set of variables of T_d. Similarly, let $Init_d(X_d)$ and $Tr_d(X_d, X'_d)$ denote the initial states and transition relation of T_d. Given a formula φ over a set of variables V, $\varphi[V \leftarrow U]$ denotes the substitution of V with U in φ. Note that $Init_d$ and Tr_d are computed from $Init$ and Tr by means of substitutions. Namely, substituting every occurrence of $x \in X$ or $x' \in X'$ with $x_d \in X_d$ and $x'_d \in X'_d$, respectively. Formally, $Init_d := Init[X \leftarrow X_d]$ and $Tr_d[X \leftarrow X_d, X' \leftarrow X'_d]$. We formulate information flow over a self-composed program as a safety verification problem: $M_d := \langle Z, Init_d, Tr_d, Bad_d \rangle$ where $Z := X \cup X_d$ and

$$Init_d(Z) := Init(X) \wedge Init(X_d) \wedge \left(\bigwedge_{x \in L} x = x_d \right) \tag{7}$$

$$Tr_d(Z, Z') := Tr(X, X') \wedge Tr(X_d, X'_d) \tag{8}$$

$$Bad_d(Z) := \left(\bigvee_{x \in L} Obs_x(X) \wedge Obs_x(X_d) \wedge \neg(x = x_d) \right) \tag{9}$$

In order to track information flow, variables in L_d are initialized to be equal to their counterpart in L, while variables in H_d remain unconstrained. A leak is captured by the bad states (i.e. Bad_d). More precisely, there exists a leak iff there exists an execution of M_d that results in a state where $Obs_x(X)$, $Obs_x(X_d)$ hold and $x \neq x_d$ for a low-security variable $x \in L$.

The algorithm for kIC3 is summarized in Algorithm 2. Since this version aims at information flow analysis, the definition of Bad and $Init$ are incorporated implicitly into the algorithm as shown in the **Candidate**, **Predecessor**$_0$ and **Unfold** rules.

The main difference between IC3 and kIC3 is that the latter maintains a different queue, which tracks pairs (k tuple) of states.

Input: A transition system $\langle Init(X),\ Tr(X, X')\rangle$, a set of high vars $H \subseteq X$ and low variables $L = X \backslash H$.

Output: *Secure* or *Unsecure*

Data: A cex queue $\mathcal{Q}$, where $c \in \mathcal{Q}$ is a tuple $\langle m, n, i\rangle$, m and n are a cubes over state variables, and $i \in \mathbb{N}$. A level N. A trace $F_0, F_1, \ldots$

Initially: $\mathcal{Q} = \emptyset$, $N = 0$, $F_0 = Init \wedge \bigwedge_{x \in L} x = x_d$, $\forall i > 0 \cdot F_i = \emptyset$.

repeat

- **Unreachable** If there is an $i < N$ s.t. $F_{i+1} \subseteq F_i$ **return** *Secure*.
- **Reachable** If there is an m and n s.t. $\langle m, n, 0\rangle \in \mathcal{Q}$ **return** *Unsecure*.
- **Unfold** If $F_N(X) \wedge F_N(X_d) \rightarrow \bigwedge_{x \in L} x = x_d$, then set $N \leftarrow N + 1$.
- **Candidate** If for some m and n, $m(X) \wedge n(X_d) \rightarrow F_N(X) \wedge F_N(X_d) \wedge \bigvee_{x \in L} x \neq x_d$, then add $\langle m, n[X_d \leftarrow X], N\rangle$ to $\mathcal{Q}$.
- **Predecessor** If $\langle m, n, i+1\rangle \in \mathcal{Q}$ and there are (m_0, m_1) and (n_0, n_1) s.t. $m_1 \rightarrow m$ and $n_1 \rightarrow n$, $m_0 \wedge m_1'$ and $n_0 \wedge n_1'$ are satisfiable, and $m_0 \wedge m_1' \rightarrow F_i \wedge Tr \wedge m'$ and $n_0 \wedge n_1' \rightarrow F_i \wedge Tr \wedge n'$, then add $\langle m_0, n_0, i\rangle$ to $\mathcal{Q}$.
- **Predecessor$_0$** If $\langle m, n, 1\rangle \in \mathcal{Q}$ and there are (m_0, m_1) and (n_0, n_1) s.t. $m_1 \rightarrow m$ and $n_1 \rightarrow n$, $m_0 \wedge m_1'$ and $n_0 \wedge n_1'$ are satisfiable, and $m_0 \wedge m_1' \rightarrow F_0 \wedge Tr \wedge m'$ and $n_0 \wedge n_1' \rightarrow F_0 \wedge Tr \wedge n'$, and $m_0|_L = n_0|_L$, then add $\langle m_0, n_0, 0\rangle$ to $\mathcal{Q}$.
- **NewLemma** For $0 \leq i < N$: given a candidate model $\langle m, n, i+1\rangle \in \mathcal{Q}$ and clause φ, such that $(\varphi \rightarrow \neg m) \vee (\varphi \rightarrow \neg n)$, if $Init \rightarrow \varphi$, and $\varphi \wedge F_i \wedge Tr \rightarrow \varphi'$, then add φ to F_j, for $j \leq i + 1$.
- **ReQueue** If $\langle m, n, i\rangle \in \mathcal{Q}$, $0 < i < N$ and $F_{i-1} \wedge Tr \wedge (m \vee n)'$ is unsatisfiable, then add $\langle m, n, i+1\rangle$ to $\mathcal{Q}$.
- **Push** For $0 \leq i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $Init \rightarrow \varphi$ and $\varphi \wedge F_i \wedge Tr \rightarrow \varphi'$, then add φ to F_j, for each $j \leq i + 1$.

until ∞;

Algorithm 2: kIC3.