Property Directed Abstract Interpretation

Noam Rinetzky[1] and Sharon Shoham[2]

[1] Tel Aviv University, Israel
[2] The Academic College of Tel Aviv Yaffo, Israel

Abstract. Recently, Bradley proposed the PDR/IC3 model checking algorithm for verifying safety properties, where forward and backward reachability analyses are intertwined, and guide each other. Many variants of Bradley's original algorithm have been developed and successfully applied to both hardware and software verification. However, these algorithms have been presented in an operational manner, in disconnect with the rich literature concerning the theoretical foundation of static analysis formulated by abstract interpretation.
Inspired by PDR, we develop a nonstandard semantics which computes for every $0 \leq N$ an over-approximation of the set of traces of length N leading to a safety violation. The over approximation is *precise*, in the sense that it only includes traces that *do not* start at an initial state, unless the program is unsafe, and in this case the semantics aborts at a special error state. In a way, the semantics computes multiple over-approximations of bounded unsafe program behaviors using a sequence of abstractions whose precision grows automatically with N.
We show that existing PDR algorithms can be described as a specific implementation of our semantics, performing an abstract interpretation of the program, but instead of aiming for a fixpoint, they stop *early* when either the backward analysis finds a counterexample or the states comprising one of the bounded traces provides sufficient evidence that the program is safe. This places PDR within the solid framework of abstract interpretation, and thus provides a unified explanation of the different PDR algorithms as well as a new proof of their soundness.

1 Introduction

Abstract interpretation [6] (*AI*) provides a solid theoretical foundation for static program analysis. *AI* algorithms verify that a program is safe by computing an over-approximation of its concrete semantics: They find a conservative representation of either the set of *reachable traces*, i.e., the traces that the program generates when executing from a given set of *initial states* (forward analysis), or of the set of *evil traces*, i.e., the ones that end in a *bad* state (backward analysis). Using the *AI* framework to develop program analyses is attractive because it elucidates the key semantic properties of the underlying abstraction and ensures, by construction, that the analysis is sound.

Recently, Bradley proposed the *property directed reachability* (PDR/IC3) model checking algorithm for verifying safety properties [3], where forward and backward analyses are intertwined, and guide each other. Many variants of Bradley's original algorithm have been developed and successfully applied to both hardware and software verification [1, 2, 5, 7, 9, 10]. However, these algorithms have been presented in an operational manner, in disconnect with the rich literature concerning abstract interpretation.

As a result, it is hard to understand and compare these algorithms without delving into minute, almost implementation-level, details.

In this paper, we provide a fresh view of the emerging family of property directed reachability verification algorithms using abstract interpretation.[3] We begin by developing an abstract trace semantics which conservatively represents the set of evil traces of length N by a sequence ω_N of sets of states, called *cartesian trace*. A cartesian trace abstracts a set of traces T by "forgetting" the fine-grained correlation between consecutive states. Cartesian traces are then further abstracted into sequences $\omega^\sharp_N$ where every set $\omega^\sharp_N(i)$ may include, in addition to the states that lead to a violation in i steps, states which are *not* reachable in $N - i$ or less steps.[4] This form of abstraction ensures that $\omega^\sharp_N$ does not represent counterexamples of length N. Furthermore, if for some N and $i < N$, it holds that $\omega^\sharp_N(i) = \omega^\sharp_N(i+1)$ then the program is safe.

In a way, our semantics can be seen as an approach to compute a conservative over approximation of the set of states leading to a safety violation, where each sequence $\omega^\sharp_N$ corresponds to a different abstraction whose precision increases automatically as N grows. The semantics considers abstract cartesian traces of every possible length simultaneously. As such, it considers infinitely many abstractions with varying precision.

An important property of our semantics is that it can capture all the *useful* fixpoints of the traditional collecting state semantics, where a fixpoint of a backward semantics is useful if it is disjoint from the set of initial states, and dually, a fixpoint of a forward semantics is useful if it is disjoint from the set of bad states.

We then use our semantics to provide a unified view of existing *PDR* algorithms: We show that they can be formulated as a specific scheduling of the semantics which stops *early* when either a counterexample is found or the program is determined to be safe. Informally, the algorithms combine backward analysis to compute (a conservative over approximation) of ω_N which are then generalized to $\omega^\sharp_N$ using forward analysis. As the formulation in terms of the semantics reveals, these algorithms consider (finitely many) cartesian traces of multiple lengths simultaneously. This places *PDR* within the solid framework of abstract interpretation, and thus presents a unified explanation of the different PDR algorithms and a new proof of their soundness.

2 Preliminaries

Binary relations. Let $R \subseteq X \times X$ be a *binary relation* over X. We write $x \xrightarrow{R} x'$ to denote that $(x, x') \in R$. We denote the *inverse* relation of R by $\overleftarrow{R}$, i.e., $\overleftarrow{R} = \{(x', x) \mid (x, x') \in R\}$. We denote the sets of elements *preceding* and *following* an element $x \in X$ according to R by $\overleftarrow{R}(x)$ and $R(x)$, respectively, i.e., $\overleftarrow{R}(x) = \{x_0 \in X \mid x_0 \xrightarrow{R} x\}$ and $R(x) = \{x' \in X \mid x \xrightarrow{R} x'\}$.

[3] In this paper, we focus on *linear* property directed reachability, as opposed to, e.g., tree-IC3 [5]. See Section 9.

[4] In model checking nomenclature, the abstraction of $\omega_N(i)$ into $\omega^\sharp_N(i)$ is called *generalization*.

We lift $R(\cdot)$, and $\overleftarrow{R}(\cdot)$ to sets in a point-wise manner, e.g., $R(X) = \{x_0 \in R(x) \mid x \in X\}$. We write $R^k(\cdot)$, and $\overleftarrow{R}^k(\cdot)$ to denote k applications of $R(\cdot)$, and $\overleftarrow{R}(\cdot)$, respectively. For example, $R^0(X) = X$ and $R^{k+1}(X) = R(R^k(X))$.

Sequences. Given a natural number $N \in \mathbb{N}$, we denote by $[N]$ the set of natural numbers from 0 to N, i.e., $[N] = \{n \in \mathbb{N} \mid 0 \leq n \leq N\}$. A *sequence* s over a set X is a total function from $[N]$, for some $N \in \mathbb{N}$, to X, i.e., $s \in [N] \to X$. We denote the set of sequences over X (including the empty sequence), by $\mathrm{seq}(X)$. We denote the length of a sequence s by $|s|$ and its i-th element by $s(i)$. For example, $s(0)$ and $s(|s|-1)$ denote the first and last elements of s, respectively. We denote the *domain* of a sequence s by $\mathrm{dom}(s)$ and its *range* by $\mathrm{range}(s)$, i.e., $\mathrm{dom}(s) = [|s|-1]$ and $\mathrm{range}(s) = \{s(i) | i \in \mathrm{dom}(s)\}$. We denote the concatenation of sequences by juxtaposition. By abuse of notation, we sometimes treat an element $x \in X$ as the sequence $\langle x \rangle$. We denote the set of sequences comprised of single elements of X by $\langle X \rangle$, i.e., $\langle X \rangle = \{\langle x \rangle \mid x \in X\}$. Let R be a binary relation. A sequence s is a *valid sequence* of R if for every $i \in [|s|-2]$, $s(i) \xrightarrow{R} s(i+1)$.

Stuttering simulation. Let X and Y be sets and $R_X \subseteq X \times X$ and $R_Y \subseteq Y \times Y$ be binary relations over X and Y, respectively. A binary relation $sim \subseteq X \times Y$ is a *stuttering simulation relation* with respect to R_X and R_Y if for every $(x, x') \in R_X$ and $(x, y) \in sim$ there exists a valid sequence of R_Y which starts at y and ends in some element $y' \in Y$ such that $(x', y') \in sim$.

States. We assume a given set of *states* Σ, ranged over by the meta-variable σ.

Transition relations and traces. We use *transition relations* and *traces* as synonyms for binary relations and sequences, respectively, when semantic elements are involved. We denote the set of transitions over states by $\Delta = \Sigma \times \Sigma$, and the set of traces over states by $\Pi = \mathrm{seq}(\Sigma)$, and range over it using π. We say that a trace is a *valid trace* of a transition relation TR if it is a valid sequence of TR. We denote the set of valid traces of TR by $[\![TR]\!]_\Pi$.

Programs and Properties. We do not commit ourselves to a particular syntax. Instead, given a *program* P, we expect to get its denotation $\mathrm{TR}(P) \subseteq \Delta$ as a transition relation over states. Similarly, we equate properties with their denotation as sets of states.

Verification problems. A *verification problem* $\mathcal{V}$ is a triple $\mathcal{V} = (Init, P, Bad)$ comprised of a set of *initial states* $Init \subseteq \Sigma$, a program P, and a set of *bad states* $Bad \subseteq \Sigma$ which does not contain initial states, i.e., $Init \cap Bad = \emptyset$. Informally, P is *safe* according to $\mathcal{V}$ if it cannot start executing in an initial state and end up in a bad state.

Conventions. In the rest of the paper, we assume a fixed arbitrary program P whose transition relation is $TR = \mathrm{TR}(P)$ and a fixed arbitrary verification problem $\mathcal{V} = (Init, P, Bad)$. Thus, whenever we say *the program*, an *initial state*, or a *bad state*, we mean P, a state in $Init$, and a state in Bad, respectively.

3 Small Step Collecting Trace Semantics

In this section, we define a small-step operational semantics over *sets* of traces.

Trace semantics. Our venture point is a rather mundane *trace semantics*, which defines the meaning of a program to be the set of traces it can produce. A trace π is a *forward trace of* P if it is a valid trace of its transition relation, i.e., if $\pi \in [\![\mathrm{TR}(P)]\!]_\Pi$. Similarly, π is a *backward trace of* P if $\pi \in [\![\overleftarrow{\mathrm{TR}(P)}]\!]_\Pi$. We say that a forward trace π of P is *reachable* if it starts in an initial state and that a backward trace π of P is *evil* if it begins in a bad state. We denote P's reachable and evil traces by $[\![P]\!]^F_\Pi$ and $[\![P]\!]^B_\Pi$, respectively:
$[\![P]\!]^F_\Pi \stackrel{\text{def}}{=} \{\pi \in [\![\mathrm{TR}(P)]\!]_\Pi \mid \pi_0 \in \mathit{Init}\}$, and $[\![P]\!]^B_\Pi \stackrel{\text{def}}{=} \{\pi \in [\![\overleftarrow{\mathrm{TR}(P)}]\!]_\Pi \mid \pi_0 \in \mathit{Bad}\}$.

Note that evil (backward) traces are read from left-to-right with the leftmost state being a bad state. And thus the backward trace transition relation $\mathrm{TR}^B_\Pi(P)$ used to define $C^B_\Pi(T)$ is in fact adding "pre-states" on the right.

We lift P's transition relation to *forward* and *backward trace transition relations*, denoted by $\mathrm{TR}^F_\Pi(P)$ and $\mathrm{TR}^B_\Pi(P)$, respectively:

$$\mathrm{TR}^F_\Pi(P) \stackrel{\text{def}}{=} \{(\pi\sigma, \pi\sigma\sigma') \mid \sigma \xrightarrow{TR} \sigma'\} \quad \text{, and} \quad \mathrm{TR}^B_\Pi(P) \stackrel{\text{def}}{=} \{(\pi\sigma, \pi\sigma\sigma') \mid \sigma \xrightarrow{\overleftarrow{TR}} \sigma'\}.$$

A trace π is reachable if there exists a valid sequence of P's forward trace transition relation leading from $\langle\sigma\rangle$ to π, where σ is an initial state. Similarly, π is evil if it is at the end of a valid sequence of the backward trace transitions starting at a trace comprised of a bad state. This allows an characterizing $[\![P]\!]^F_\Pi$ and $[\![P]\!]^B_\Pi$ as least fixpoints:

$$\begin{aligned}
[\![P]\!]^F_\Pi &= \mathrm{LFP}\ C^F_\Pi \quad \text{where} \quad C^F_\Pi(T) = \langle \mathit{Init}\rangle \cup \mathrm{TR}^F_\Pi(P)(T) \quad \text{, and} \\
[\![P]\!]^B_\Pi &= \mathrm{LFP}\ C^B_\Pi \quad \text{where} \quad C^B_\Pi(T) = \langle \mathit{Bad}\rangle \cup \mathrm{TR}^B_\Pi(P)(T).
\end{aligned}$$

Small step collecting trace semantics. C^F_Π and C^B_Π, defined above, operate on sets of traces. Such sets are in fact elements of the *collecting trace semantics* of P. The latter interprets P by accumulating the traces generated by its trace semantics. Formally, the *collecting trace domain* $\mathcal{D}_\Pi = (\mathscr{P}(\Pi), \subseteq)$ is a powerset domain over the set of traces, ordered by set inclusion.

A collecting semantics is often used as means to compute fixpoints of an underlying operational semantics. However, it can also be given an operation flavor by defining initial sets of traces and transitions between sets of traces. The initial set of traces is $\langle \mathit{Init}\rangle$ in the forward collecting trace semantics, and $\langle \mathit{Bad}\rangle$ in the backward semantics. The transitions are defined as the pointwise lifting of P's forward and backward trace transition relation to sets of traces, denoted by $\mathrm{TR}^F_{\mathscr{P}(\Pi)}(P)$ and $\mathrm{TR}^B_{\mathscr{P}(\Pi)}(P)$, respectively:

$$\begin{aligned}
\mathrm{TR}^F_{\mathscr{P}(\Pi)}(P) &\stackrel{\text{def}}{=} \{(T, T \cup \{\pi'\}) \mid \exists \pi \in T.\, \pi \xrightarrow{\mathrm{TR}^F_\Pi(P)} \pi'\} \text{ , and} \\
\mathrm{TR}^B_{\mathscr{P}(\Pi)}(P) &\stackrel{\text{def}}{=} \{(T, T \cup \{\pi'\}) \mid \exists \pi \in T.\, \pi \xrightarrow{\mathrm{TR}^B_\Pi(P)} \pi'\}.
\end{aligned}$$

Note that both $[\![P]\!]^F_\Pi$ and $[\![P]\!]^B_\Pi$ are elements of $\mathcal{D}_\Pi$. Recall that we consider only finite sequences. Thus, there might not be a valid sequence according to, e.g., $\mathrm{TR}^B_{\mathscr{P}(\Pi)}(P)$ which leads from from $\langle \mathit{Bad}\rangle$ to $[\![P]\!]^B_\Pi$ because $[\![P]\!]^B_\Pi \setminus \langle \mathit{Bad}\rangle$ might be an infinite set. However, for every set of traces $T \supseteq \langle \mathit{Bad}\rangle$ and every *finite* set of evil traces T', there is such a valid sequence going from T to T'. Formally:

Lemma 1. *For every trace π, it holds that $\pi \in [\![P]\!]^F_\Pi$ (respectively, $\pi \in [\![P]\!]^B_\Pi$) if and only if there is a valid sequence of $\mathrm{TR}^F_{\mathscr{P}(\Pi)}(P)$ (respectively, $\mathrm{TR}^B_{\mathscr{P}(\Pi)}(P)$) going from $\langle \mathit{Init}\rangle$ (respectively, $\langle \mathit{Bad}\rangle$) to T such that $\pi \in T$.*

In Section 8, we show that *PDR* can be formalized as an abstract interpretation of a program using (a conservative abstraction of) the collecting trace semantics which develops simultaneously multiple traces. However, instead of trying to compute a fixpoint of the program's collecting trace semantics, *PDR* uses the execution as means to come up with a *useful fixpoint* of its *collecting state semantics* as we explain below.

Collecting state semantics. It is standard to abstract a set of traces by the set of their states. Formally, the *collecting state semantics* of programs is a powerset domain over the set of states, ordered by set inclusion $\mathcal{D}_\Sigma = (\mathscr{P}(\Sigma), \subseteq)$. We define the expected Galois connection $(\mathcal{D}_\Pi, \alpha_\Sigma, \gamma_\Sigma, \mathcal{D}_\Sigma)$ between sets of traces and sets of states:

$$\begin{array}{lll} \gamma_\Sigma : \mathscr{P}(\Sigma) \to \mathscr{P}(\Pi) & ::= & \gamma_\Sigma(S) = \{\pi \in \Pi \mid \forall i \in \mathrm{dom}(\pi).\, \pi(i) \in S\} \text{ and} \\ \alpha_\Sigma : \mathscr{P}(\Pi) \to \mathscr{P}(\Sigma) & ::= & \alpha_\Sigma(T) = \{\sigma \in \mathrm{range}(\pi) \mid \pi \in T\}\,. \end{array}$$

We say that a state is *reachable* if it appears in a reachable trace and *evil* if it appears in an evil one. The sets of reachable and evil states, denoted by $[\![P]\!]^F_\Sigma$ and $[\![P]\!]^B_\Sigma$, respectively, are defined using abstraction, and enjoy a least fixpoint characterization:

$$\begin{array}{lllll} [\![P]\!]^F_\Sigma \stackrel{\text{def}}{=} \alpha_\Sigma([\![P]\!]^F_\Pi) & = & \mathrm{LFP}\ C^F_\Sigma & \text{where} & C^F_\Sigma(S) = \mathit{Init} \cup \mathit{TR}(S) \quad \text{, and} \\ [\![P]\!]^B_\Sigma \stackrel{\text{def}}{=} \alpha_\Sigma([\![P]\!]^B_\Pi) & = & \mathrm{LFP}\ C^B_\Sigma & \text{where} & C^B_\Sigma(S) = \mathit{Bad} \cup \overleftarrow{\mathit{TR}}(S)\,. \end{array}$$

4 Useful and Projected Fixpoints

We refer to an evil trace that leads to an initial state as a *counterexample*. A program P is *safe* if none of its evil traces is a counterexample, and *unsafe* otherwise. In our setting, safety amounts to requiring that $[\![P]\!]^F_\Sigma \cap [\![P]\!]^B_\Sigma = \emptyset$. The goal of *PDR* and of its variants is to compute a superset of the reachable states of P, if P is safe, and report that a counterexample exists, otherwise. This is often done by looking for an inductive fixpoint of the (forward or backward) collecting state semantics.

A set of states S is an *inductive (forward) fixpoint* if $\mathit{Init} \subseteq S$ and $\mathit{TR}(S) \subseteq S$. We say that S is a *useful (forward) fixpoint* if, in addition, $S \cap \mathit{Bad} = \emptyset$. (A useful forward fixpoint is often called a *safe inductive invariant*.) Similarly, S is an *inductive backward fixpoint* if $\mathit{Bad} \subseteq S$ and $\overleftarrow{\mathit{TR}}(S) \subseteq S$. It is *useful* if $S \cap \mathit{Init} = \emptyset$.

A standard technique to find an inductive fixpoint is to iteratively apply the corresponding transformer. For example, to find an inductive fixpoint of the backward collecting state semantics, we would usually repeatedly apply C^B_Σ, while accumulating the discovered states, until no new state is discovered. As C^B_Σ is monotonic, it is ensured by Kleene's Theorem that at the limit we reach its least fixpoint. However, we can find such a fixpoint in a different way via a *projection* of the elements computed by the collecting trace semantics using, what we refer to, as *projected fixpoints*.

Given a set of traces T, we denote by $T|^i_\Sigma = \{\pi(i) \mid \pi \in T \wedge i < |\pi|\}$ the set of states in the i-th index of the traces in T. If there exists an index $i > 0$ such that (i) $\mathit{Bad} \subseteq T|^0_\Sigma$, (ii) for every $0 \leq j \leq i$, $\overleftarrow{\mathit{TR}}(T|^j_\Sigma) \subseteq T|^{j+1}_\Sigma$, and (iii) $T|^i_\Sigma = T|^{i+1}_\Sigma$, then $S = \bigcup_{j=0}^{i} T|^j_\Sigma$ is an inductive backward fixpoint of the collecting state semantics. We refer to S as a *projected fixpoint* of the collecting trace semantics. Intuitively, every evil trace can go only through states that appear in S. We note that if T has been computed by accumulating the results of some $0 \leq k$ applications of $C^B_\Pi(\cdot)$ starting from $\emptyset$, it suffices to check point (iii) above to determine that T has a projected fixpoint.

5 Small Step Cartesian Trace Semantics

The *cartesian trace semantics* abstracts the (forward and backward) collecting trace semantics using sequences of sets of states, which we refer to as *cartesian traces*. Informally, a cartesian trace ω conservatively represents a set of traces T of length $|\omega|$ or less by abstracting away the correlation between consecutive states. In the following, we focus on abstracting the backward semantics, as it is the one used by ***PDR***. The cartesian semantics is suitable for tracking the intermediate results that occur during an iterative conservative fixpoint computation, and thus fits well to describe the sequence of sets of states computed by ***PDR***. We refer to the set components of cartesian traces as *anti-frames*, as they correspond to the complements of the sets maintianed by ***PDR***, which are often referred to as *frames*. (See Section 7.)

5.1 Cartesian Trace Transition Relation

We denote by $\Omega = \mathrm{seq}(\mathscr{P}(\Sigma))$ the set of all sequences of sets of states, ranged over by metavariable ω. Following the intuitive discussion above, we define a function γ_ω which maps a cartesian trace ω to the set of traces that it represents. The latter is comprised of any trace whose i-th state, for every i, is taken from the corresponding set $\omega(i)$.

$$\gamma_\omega : \Omega \to \mathscr{P}(\Pi) \quad ::= \quad \gamma_\omega(\omega) \stackrel{\text{def}}{=} \{\pi \in \Pi \mid |\pi| \leq |\omega| \wedge \forall i \in \mathrm{dom}(\pi).\, \pi(i) \in \omega(i)\}\,.$$

Note that if ω represents a trace π, then ω also represents every prefix of π.

Cartesian traces allow to *over-approximate* the (backward) trace semantics of P by lifting P's transition relation to a *backward cartesian trace transition relation*, denoted by $\mathrm{TR}^B_\Omega(P)$:

$$\mathrm{TR}^B_\Omega(P) \stackrel{\text{def}}{=} \{(\omega_1 S_1 S_2 \omega_2,\, \omega_1 S_1 (S_2 \cup S) \omega_2) \mid S \subseteq \overleftarrow{TR}(S_1)\}\,.$$

Note that while the collecting transition relation $\mathrm{TR}^B_{\mathscr{P}(\Pi)}(P)$ extends traces, the cartesian transition relation relates only traces of the same length. Indeed, it can only add new states to sets that ω already contains. Intuitively, this means that we can only over-approximate at most $|\omega|-1$ consecutive transitions of P. We do not overcome this limitation, instead we weaken the guarantees we get from abstract interpretation of P according to the cartesian trace semantics, as we shortly explain.

5.2 Cartesian Traces Domain

To define the *cartesian traces domain*, we first introduce the *subsumption* order between cartesian traces. We say that ω_1 *subsumes* ω_2, denoted by $\omega_1 \preceq_\omega \omega_2$, if every entry in ω_2 subsumes the corresponding entry in ω_1. Formally,

$$\omega_1 \preceq_\omega \omega_2 \stackrel{\text{def}}{=} |\omega_1| = |\omega_2| \wedge \forall i \in \mathrm{dom}(\omega_1).\, \omega_1(i) \subseteq \omega_2(i)\,,$$

The cartesian traces domain $\mathcal{D}_\Omega = (\mathscr{P}(\Omega), \sqsubseteq_\Omega)$ utilizes the powerset of the cartesian traces as its carrier set and it is ordered by a point-wise lifting of subsumption:

$$\mathcal{D}_\Omega = (\mathscr{P}(\Omega), \sqsubseteq_\Omega)\text{ , where } O_1 \sqsubseteq_\Omega O_2 \iff \forall \omega_1 \in O_1.\, \exists \omega_2 \in O_2.\, \omega_1 \preceq_\omega \omega_2\,.$$

The Galois connection $(\mathcal{D}_\Pi, \alpha_\Omega, \gamma_\Omega, \mathcal{D}_\Omega)$ between the domain of traces and that of cartesian traces is defined by a pointwise lifting of γ_ω to sets of cartesian traces.

$$\begin{array}{lll} \gamma_\Omega : \mathscr{P}(\Omega) \to \mathscr{P}(\Pi) & ::= & \gamma_\Omega(O) \;=\; \{\pi \in \gamma_\omega(\omega) \mid \omega \in O\} \text{ , and} \\ \alpha_\Omega : \mathscr{P}(\Pi) \to \mathscr{P}(\Omega) & ::= & \alpha_\Omega(T) \;=\; \{\lambda i \in \mathrm{dom}(\pi).\, \{\pi(i)\} \mid \pi \in T\}\,. \end{array}$$

Lemma 2. *$(\mathcal{D}_\Pi, \alpha_\Omega, \gamma_\Omega, \mathcal{D}_\Omega)$ is a Galois connection.*

Lemma 3. *Let π be a trace and ω be a cartesian trace such that $0 < |\pi| < |\omega|$. If $\pi \in \gamma_\omega(\omega)$ then $\mathrm{TR}^B_\Pi(P)(\pi) \subseteq \gamma_\Omega(\mathrm{TR}^B_\Omega(P)(\omega))$.*

Lemma 3 ensures that $\mathrm{TR}^B_\Omega(P)(\cdot)$ is a sound abstract transformer with respect to $\mathrm{TR}^B_\Pi(P)(\cdot)$ when we consider only bounded executions. More specifically, given a cartesian trace ω of length n and a trace π of length m represented by ω, we can over-approximate the set of traces that can be reached by executing $n-m-1$ trace transitions $\xrightarrow{\mathrm{TR}^B_\Pi(P)}$. In particular, if $\omega(0) = Bad$, we can use $\mathrm{TR}^B_\Omega(P)(\cdot)$ to over-approximate the evil traces of length n or less.

In a sense, the cartesian trace semantics allows to over-approximate bounded under-approximations of the standard collecting trace semantics.

6 Property-Guided Abstraction of the Cartesian Trace Semantics

We abstract the backward cartesian trace semantics in a property-guided manner using two means: Firstly, we go to an *error state* in case we find a counterexample. Secondly, and most importantly, we allow to over-approximate the backward cartesian transition relation in a controlled way which ensures that the abstract trace does not represent spurious counterexamples. This form of abstraction explains the *generalization* operations in *PDR*. (See Section 7).

6.1 Property-Guided Cartesian Trace Transition Relation

The property-guided cartesian trace semantics over-approximates the backward cartesian trace transition relation by adding two new kinds transitions: *generalization* transitions, denoted by $\mathrm{TR}^{Gen(B)}_\Omega(P)$, and *error* transitions, denoted by $\mathrm{TR}^{Err(B)}_\Omega(P)$, which lead to a special *error* element $\top$.

$$\begin{array}{lll} \mathrm{TR}^{Gen(B)}_\Omega(P) & \stackrel{\text{def}}{=} & \{(\omega_1 S_1 S_2 \omega_2,\, \omega_1(S_1 \cup Y) S_2 \omega_2) \mid \overleftarrow{TR}(Y) \subseteq S_2 \wedge Y \cap \mathit{Init} = \emptyset\} \quad \text{, and} \\ \mathrm{TR}^{Err(B)}_\Omega(P) & \stackrel{\text{def}}{=} & \{(\omega,\, \top) \mid \omega(|\omega|-1) \cap \mathit{Init} \neq \emptyset\}\,. \end{array}$$

Generalization transitions add a "forward" flavor to the property-guided cartesian trace semantics as they add states at index j based on the states at index $j+1$. (Recall that these are *backward* traces, hence updates of $j+1$ based on j correspond to backward steps, while updates of j based on $j+1$ correspond to forward steps.)

Given a cartesian trace $\omega = \omega_1 S_1 S_2 \omega_2$, a generalization transition allows to add to its j-th anti-frame, where $j = |\omega_1|$, any state σ such that any backward trace of P

of length $|\omega|-j$ which starts at σ goes only through states that can be reached by a backward trace starting at one of the states in the $j+1$ anti-frame. Thus, the states added by the generalization would not open a new route towards an undiscovered state. Specifically, generalization would not lead to over-approximating a counterexample, unless this counterexample is already represented.

An error transition, happens when we find an initial state at the last anti-frame of the trace. Note that this means that we found a counterexample. It suffices to look for an initial state *only* in the last anti-frame because of our assumption that $Init$ and Bad are disjoint and the restrictions on the transformers which ensure that if the semantics computes a trace which goes through an initial state, it can also compute a shorter (evil) trace which ends with that state.

We denote the enriched transition relation by $\mathrm{TR}_{\Omega}^{BGE}(P)$, i.e.,

$$\mathrm{TR}_{\Omega}^{BGE}(P) = \mathrm{TR}_{\Omega}^{B}(P) \cup \mathrm{TR}_{\Omega}^{Gen(B)}(P) \cup \mathrm{TR}_{\Omega}^{Err(B)}(P)\,.$$

In the following, we refer to the transitions defined in Section 5.1 as *pre-transitions*. We say that a pre-transition $(\omega_1 S_1 S_2 \omega_2,\, \omega_1 S_1 (S_2 \cup S)\omega_2) \in \mathrm{TR}_{\Omega}^{B}(P)$ *takes place* at index $|\omega_1|$. (Note that we say that although the transition updates the set at index $|\omega_1|+1$). We say that a gen-transition $(\omega_1 S_1 S_2 \omega_2,\, \omega_1 (S_1 \cup Y) S_2 \omega_2) \in \mathrm{TR}_{\Omega}^{Gen(B)}(P)$ *takes place* at index $|\omega_1|$ based on the set at index $|\omega_1|+1$.

6.2 Small Step Collecting Property-Guided Cartesian Trace Semantics

Recall that the cartesian transition relation does not allow to extend the length of a trace ω, nor do the generalization and error transition relations, and hence they are limited to over-approximate bounded executions. To overcome this limitation, we turn to the powerset domain; the underlying domain of the collecting property-guided cartesian trace semantics is the cartesian trace domain, $\mathcal{D}_{\Omega}$, enriched with the error element, $\top$, which is greater than any other element.

Property-guided initial cartesian traces. We prepare ahead to produce traces of any possible length by starting the interpretation of the program from an unbounded set of cartesian traces: Let $\emptyset^k$ denote a cartesian trace of length $0 \leq k$ whose anti-frames are all empty, i.e., $\emptyset^k = \langle \emptyset, \ldots, \emptyset \rangle$. A cartesian trace ω is *property-guided initial* (*initial* for short) if $\omega = \langle Bad \rangle \emptyset^k \langle \Sigma \setminus Init \rangle$, for some $0 \leq k$, i.e., its first anti-frame is comprised of bad states, its last of the non-initial ones, and all the others are empty. Note that all initial cartesian traces are of length ≥ 2.

We denote the *initial* cartesian trace of length i (for $i \geq 2$) by $\hat{\omega}^i$, i.e., $\hat{\omega}^i = \langle Bad \rangle \emptyset^{i-2} \langle \Sigma \setminus Init \rangle$, and the set of initial cartesian traces by $\hat{\Omega}$. Note that $\hat{\omega}^2$ represents all traces of length 2 that start in a bad state and end in a non-initial state as well as their prefixes, i.e., if the program is safe $\hat{\omega}^2$ represents the largest safe over-approximation (superset) of the evil traces of P of length at most two. All other initial cartesian traces represent $\langle Bad \rangle$, the set of evil traces of length one (which correspond to the prefix of length one since the second element is $\emptyset$). Informally, starting from a given initial cartesian trace $\hat{\omega}^i$, we can simulate evil traces of length i or less.

Property-guided collecting cartesian transition relation. The *collecting property-guided cartesian trace semantics* is obtained by lifting the enriched transition relation

$\mathrm{TR}^{BGE}_{\Omega}(P)$ to a collecting transition relation $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$ which works in a pointwise manner on sets of cartesian traces. This is done similarly to the way we obtained the transition relation of the collecting trace semantics $\mathrm{TR}^{B}_{\mathscr{P}(\Pi)}(P)$ out of that of the trace semantics $\mathrm{TR}^{B}_{\Pi}(P)$. (See, Section 3.) We also adapt $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)(O)$ to go to $\top$ if there is a cartesian trace in O that leads to $\top$ in one step. The valid sequences of $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$ from $\hat{\Omega}$ define the property-guided meaning of the program.

Lemma 4 (Soundness and Precision). *A program P is safe if and only if for any $0 \leq k$, it holds that $(\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P))^k(\hat{\Omega}) \neq \top$.*

Lemma 4 ensures that we can use the property-guided cartesian trace semantics to find any evil trace of P. Intuitively, we can compute any evil trace π by first picking an initial cartesian trace of length $|\pi|+1$ and then executing the sequence of cartesian trace transitions corresponding to the ones which generated π. Furthermore, it ensures that the property-guided semantics does not lose precision when it comes to safety: Thanks to the restrictions on the generalization steps, the semantics never reaches an error state if the program is safe.

We can adapt the notion of projected fixpoints to the cartesian semantics. Given a cartesian trace ω, we say that $\omega(i)$, where $0 < i < |\omega|-1$, is a *projected fixpoint* if (i) $Bad \subseteq \omega(0)$, (ii) for every $0 \leq j < i$, $\omega(j) \cup \overleftarrow{TR}(\omega(j)) \subseteq \omega(j+1)$, and (iii) $\omega(i) = \omega(i+1)$.

Lemma 5 (Projected fixpoints). *Let ω be a cartesian trace such that $\omega(i)$ is a projected fixpoint. It holds that $\omega(i)$ is an inductive backward fixpoint of the collecting state semantics.*

In fact, given a useful backward fixpoint S, we can use the appropriate generalization transitions starting from $\hat{\Omega}$ to produce a cartesian trace ω which contains S as a projected fixpoint at some index i.

We can now restate the last paragraph of Section 3 in a more precise way: In Section 8, we show that *PDR* can be formalized as an abstract interpretation of the collecting property-guided cartesian semantics, where every operation of *PDR* can be understood as a sequence of steps taken by the semantics.

The semantics, when looking at it from the viewpoint of *PDR*, interprets the program with two goals in mind. The first goal is to look for a *useful fixpoint* of its *collecting state semantics*. This is done by taking generalization steps. The second goal, which is done in parallel, is to look for a counterexample. This is done using pre-transitions. The two goals affect each other: The states that are discovered using the pre-transitions, are used to compute Y in the generalization transitions by applying an algorithm specific-heuristic. The generalization, on the other hand, might add states that would make future pre-transitions mute as their targets would be detected early. This, could help *PDR* terminate faster than if it had taken only pre-transitions.

The *PDR*-viewpoint helps understand the reason behind placing $\Sigma \setminus \mathit{Init}$ as the last component of the initial cartesian traces: It is apriori known that this set provides (the most coarse) over-approximation of the last state of any evil trace which is not a

counterexample. As a result, it provides the greatest opportunity to apply generalization transitions at the penultimate set, and by extension, at the ones preceding it. This flexibility is the reason that the collecting semantics can compute any useful fixpoint.

7 Traditional *PDR*

In this section we describe *PDR* in an operational manner. Traditionally, *PDR* uses a symbolic representation of states and sets of states as formulas in some logic (either propositional or first order logic). In our description of *PDR* we refer to the underlying states or sets of states explicitly.

We start by a high-level description of *PDR* and the data structures used by it. The latter also define its *configurations*. We then describe the different operations performed by the different implementations of *PDR*.

Initially, *PDR* checks if $Init \cap Bad = \emptyset$, and reports a counterexample if this is not the case. For simplicity of the presentation, we consider this check to be done before *PDR* is invoked. We therefore assume that $Init \cap Bad = \emptyset$.

Forward reachability sequence. *PDR* computes increasingly longer *forward reachability sequences*. When referring to sequences maintained by *PDR*, we use a subscript notation for the elements of a sequence: F_i instead of $F(i)$. We denote the sequence comprised of the elements $F_0, \ldots, F_N$, for some $0 < N$, by $\langle F_0, \ldots, F_N \rangle$.

Definition 1 (Forward Reachability Sequence). *A* forward reachability sequence *of length $N+1$ is a sequence $\varphi_N = \langle F_0, F_1, \ldots, F_N \rangle \in \mathrm{seq}(\mathscr{P}(\Sigma))$ which has the following properties:*

1. $F_0 = Init$,
2. $F_i \subseteq F_{i+1}$ *for every* $0 \leq i < N$,
3. $TR(F_i) \subseteq F_{i+1}$ *for every* $0 \leq i < N$,
4. $F_i \cap Bad = \emptyset$ *for every* $0 \leq i \leq N$.

The sets F_i in the sequence φ_N are called frames. *N is called the* iteration counter.

Note that the property $TR(F_i) \subseteq F_{i+1}$ is equivalent to $\overleftarrow{TR}(\Sigma \setminus F_{i+1}) \subseteq \Sigma \setminus F_i$. We use the two interchangeably. The properties of a forward reachability sequence φ_N imply that for every $0 \leq i \leq N$, frame F_i over-approximates the set of states reachable from the initial states in at most i steps. If the sequence includes an index $0 \leq i < N$ such that $F_i = F_{i+1}$ then property 3 simplifies to $TR(F_i) \subseteq F_i$. Hence, together with properties 1 and 4, we conclude that F_i is a useful forward fixpoint (or safe inductive invariant), which implies that P is safe.

PDR computes forward reachability sequences φ_N of increasing lengths, starting from $N = 1$, until either a counterexample is found or a fixpoint is reached.

In the intermediate steps of the computation of the forward reachability sequence φ_N, requirement 3 might not hold (*only*) for $i = N - 1$, in which case we refer to φ_N as an *intermediate forward sequence*. Specifically, for $N = 1$, φ_N is initialized to $\langle Init, \Sigma \setminus Bad \rangle$. For $N > 1$, *PDR* initializes an intermediate forward sequence φ_N by extending the forward reachability sequence φ_{N-1} from the previous iteration with an additional frame $F_N = \Sigma \setminus Bad$. If requirement 3 does not hold due to the addition of F_N, *PDR* tries to strengthen the frames F_i (which over-approximate the reachable

states) in order to satisfy requirement 3 for $i = N - 1$ as well. For this purpose, *PDR* iteratively retrieves from F_{N-1} a state for which $TR(\sigma) \subseteq F_N$ does not hold (equivalently, $\sigma \in \overleftarrow{TR}(Bad) \cap F_{N-1}$), and tries to eliminate it by strengthening F_{N-1}. To do so while maintaining the (other) properties of a forward reachability sequence, *PDR* first has to strengthen F_{N-2} to eliminate from it all the predecessors of σ. For the elimination of each predecessor, the same process is needed. This results in a backward traversal of the state space.

Obligations queue. The states that need to be eliminated from their frames are called *counterexamples to induction* (CTIs), since their removal is needed in order to maintain the induction condition ($TR(F_i) \subseteq F_{i+1}$). A pair (i, σ) consisting of an index i and a CTI σ that needs to be eliminated from F_i is called a *proof obligation* (obligation in short). All obligations have the property that their states lead to a bad state. Technically, *PDR* uses an *obligation queue*, denoted q, to handle the obligations.

If all obligations are handled successfully, φ_N satisfies requirement 3 for $i = N-1$ as well, and hence it becomes a forward reachability sequence. However, there might be intermediate steps where q is temporarily empty, even though not all obligations have been handled (since not all have been discovered). To distinguish between the former and the latter we use $\perp$ to denote the value of the queue when all obligations are handled, as opposed to $\emptyset$ which denotes an empty queue, possibly temporarily.

PDR *configurations.* A configuration of *PDR* is a triple $\kappa = (N, \varphi_N, q)$, where

- $N \in \mathbb{N}$,
- $\varphi_N = \langle F_0, F_1, \ldots, F_N \rangle \in (\mathscr{P}(\Sigma))^{N+1}$ is an intermediate forward sequence, and
- $q \in \mathscr{P}([N] \times \Sigma) \cup \{\perp\}$ is an obligations queue, where $[N] = \{0, \ldots, N\}$.

Initial configuration. Assuming that $Init \cap Bad = \emptyset$, the initial configuration of *PDR* is $\kappa_0 = (1, \langle Init, \Sigma \setminus Bad \rangle, \emptyset)$.

PDR *operations.* Given a configuration $\kappa = (N, \varphi_N, q)$ as above, *PDR* proceeds by performing one of the following procedures. We denote the resulting configuration by $\kappa' = (N', \varphi', q')$. Each procedure updates a subset of the components of the configuration. We describe only the components that are indeed updated.

Queue initialization: If $q = \emptyset$, and there is a state $\sigma \in \overleftarrow{TR}(Bad) \cap F_{N-1}$, *PDR* adds the obligation $(N-1, \sigma)$ to the queue, resulting in $q' = \{(N-1, \sigma)\}$. If no such state exist, it sets $q' = \perp$.

Backward step: Given an obligation $(i, \sigma') \in q$, where $1 \leq i \leq N$ is the minimal frame index in q, such that there is $\sigma \in \overleftarrow{TR}(\sigma') \cap F_{i-1}$, *PDR* adds $(i-1, \sigma)$ to q. Namely, $q' = q \cup \{(i-1, \sigma)\}$.

Obligation lifting: Once an obligation $(i-1, \sigma)$ is added to q due to a backward step from $(i, \sigma') \in q$, *PDR* computes a *lifting* of the obligation, $S = \mathit{OLift}(\sigma, \sigma', F_i)$, and adds the set of obligations $\{i-1\} \times S$ to the queue, where $\mathit{OLift}(\sigma, \sigma', F_i)$ computes a set of states $S \subseteq \Sigma$ such that $S \subseteq \overleftarrow{TR}(\sigma')$. Namely, $q' = q \cup (\{i-1\} \times S)$.

Obligation lifting helps accelerating *PDR* by lifting an obligation discovered by a backward step from some obligation $(i, \sigma') \in q$ to a *set* of obligations, all of which result from a backward step of the same obligation.

Blocking: Given an obligation $(i, \sigma') \in q$, where $1 \leq i \leq N$ is the minimal frame index in q and $\overleftarrow{TR}(\sigma') \cap F_{i-1} = \emptyset$, ***PDR*** removes (i, σ') from q, and removes σ' from F_i (if it was not yet removed). Note that since $i \geq 1$, $\sigma' \notin \mathit{Init}$. This results in the configuration $\kappa' = (N, \langle F_0, \ldots, F_{i-1}, F_i \setminus \{\sigma'\}, F_{i+1}, \ldots, F_N \rangle, q \setminus \{(i, \sigma')\})$.

Generalization: Once (i, σ') is blocked, in addition to removing σ' from F_i, ***PDR*** computes a *generalization* of the blocked state, $S = \mathit{Gen}(\sigma', F_{i-1})$, and removes S from all F_j such that $j \leq i$, where $\mathit{Gen}(\sigma', F_{i-1})$ computes a set of states $S \subseteq \Sigma$ such that $\mathit{Init} \cap S = \emptyset$ and $TR(F_{i-1}) \cap S = \emptyset$ (i.e., where all states have no predecessor in F_{i-1}). The result is $\varphi' = \langle F_0, F_1 \setminus S, \ldots, F_i \setminus S, F_{i+1}, \ldots, F_N \rangle$.

Inductive generalization: Once (i, σ') is blocked, in addition to removing σ' from F_i, ***PDR*** computes an *inductive generalization* of the blocked state, $S = \mathit{IGen}(\sigma', F_{i-1})$, and removes S from all F_j such that $j \leq i$, where $\mathit{IGen}(\sigma', F_{i-1})$ computes a set of states $S \subseteq \Sigma$ such that $\mathit{Init} \cap S = \emptyset$ and $TR(F_{i-1} \setminus S) \cap S = \emptyset$. The result is $\varphi' = \langle F_0, F_1 \setminus S, \ldots, F_i \setminus S, F_{i+1}, \ldots, F_N \rangle$.

Inductive generalization is an enhancement of generalization which results in a stronger strengthening of frames, as every generalization is also an inductive generalization, but not the other way around. It is based on an attempt to identify sets whose complements are inductive *relatively to the current frame*, and therefore can be used to safely strengthen all frames up to the current one while keeping the properties of an intermediate forward sequence (and in particular, without excluding any reachable state).

Forward propagation: Once F_i is updated by removing S from it (as a result of generalization, inductive generalization, or forward propagation), i.e. $F_i \cap S = \emptyset$, it is checked whether $TR(F_i) \cap S = \emptyset$, and if so, F_{i+1} is also updated to $F_{i+1} \setminus S$. The result is $\varphi' = \langle F_0, \ldots, F_i, F_{i+1} \setminus S, F_{i+2}, \ldots, F_N \rangle$.

Forward propagation attempts to speculatively strengthen frames before obligations are encountered. Similarly to inductive generalization, it considers sets that are inductive relatively to the current frame (the complement of every set that is removed from a frame corresponds to such a relative inductive set), and checks whether they are also inductive relatively to consecutive frames.

Pushing obligations forward: Once an obligation (i, σ') for $1 \leq i \leq N-1$ is removed from q, an obligation $(i+1, \sigma')$ is added to q. The result is $q' = q \cup \{(i+1, \sigma')\}$.

Pushing obligations forward aims at an early discovery of obligations. An obligation (i, σ') consists of a state σ' that reaches a bad state in some $k > 0$ steps. The same holds also when σ' is considered in F_{i+1}, which makes $(i+1, \sigma')$ a legitimate obligation (it will be discovered/handled at the latest when $N = i + 1 + k$). Its early addition can help accelerate the strengthening towards a fixpoint, or enable finding counterexamples that are longer than $N + 1$.

Unfolding: If $q = \bot$ and fixpoint is not obtained, ***PDR*** initializes F_{N+1} to $\Sigma \setminus \mathit{Bad}$, increases N to $N + 1$, and sets q to an empty queue. This results in the configuration $\kappa' = (N+1, \langle F_0, F_1, \ldots, F_N, \Sigma \setminus \mathit{Bad} \rangle, \emptyset)$.

Termination. If there is an obligation $(0, \sigma') \in q$, ***PDR*** terminates and reports a *counterexample*. If $q = \bot$, and there exists $i < N$ such that $F_i = F_{i+1}$, ***PDR*** terminates with a *fixpoint* and reports safety.

PDR is parametric in the generalization function Gen, the inductive generalization function $IGen$ (typically only one of them is used), and the lifting function *OLift*.

Remark 1 (Symbolic PDR*). PDR* is typically implemented as a SAT-based or an SMT-based model checking algorithm. It uses formulas in (propositional or first order) logic over a vocabulary V to describe states and sets of states. In particular, a state is described as a *cube* over V, i.e., a conjunction of literals (predicates or their negations) and a set (e.g., a frame F_i) is described as a CNF formula over V, i.e., conjunction of clauses where each clause consists of a disjunction of literals. The transition relation TR is also described by a formula, over a double vocabulary $V \cup V'$, where V represents the current state and $V' = \{v' \mid v \in V\}$ represents the next state.

Checks such as $\overleftarrow{TR}(\sigma') \cap F_{i-1} = \emptyset$ are done by validity checks of the corresponding formulas, e.g. $F_{i-1}(V) \wedge TR(V, V') \Rightarrow \neg\sigma'(V')$, or alternatively, unsatisfiability checks of their negation, i.e., $F_{i-1}(V) \wedge TR(V, V') \wedge \sigma'(V')$. When the formula is satisfiable, a state $\sigma \in \overleftarrow{TR}(\sigma') \cap F_{i-1}$ is retrieved from the satisfying assignment.

In this setting, generalization, inductive generalization and lifting are performed on a cube, representing a state, and a CNF formula, representing a frame. They compute a CNF formula representing a set of states.

For example, a typical implementation of generalization $Gen(\sigma', F_{i-1})$ looks for a sub-clause c of the clause $\neg\sigma'(V)$ such that $Init(V) \Rightarrow c(V)$ and $F_{i-1}(V) \wedge TR(V, V') \Rightarrow c(V')$. If this holds, then $Gen(\sigma', F_{i-1})$ returns $\neg c(V)$ as a formula representing the set of states to be removed from F_j for all $j \leq i$. The removal is performed by conjoining F_j with c. Inductive generalization is performed similarly.

Obligations lifting was performed in the original *PDR* paper [3] statically by considering the k-step cone of influence. [7] performed dynamic lifting using ternary simulation. [4] suggested a SAT-based approach, using unsatisfiability cores, for lifting.

8 PDR as a Property-Guided Abstract Interpretation of the Cartesian Trace Semantics

In this section, we show that the *collecting* property-guided cartesian trace semantics defined in Section 6 simulates *PDR*, or in other words, *PDR* is an implementation of the semantics. For this purpose we define a simulation relation mapping *PDR* configurations to elements of the semantics, given by sets of sequences. We show that each step of *PDR* is simulated by a sequence of transitions of the semantics, in the sense that the resulting *PDR* configuration matches the resulting element in the semantics.

The mapping between *PDR* configurations and elements of the semantics is given by a *compatibility relation* defined below. It should be noted that while the sequences of frames used by *PDR* are indexed such that $F_0 = Init$ and increasing indices represent increasing distance (with respect to TR) from the initial states, the sequences used by our semantics are indexed such that $\omega(0) = Bad$ and increasing indices represent increasing distance (with respect to $\overleftarrow{TR}$) from the bad states. In this sense, the two consider opposite directions of the transition relation.

Definition 2 (Compatibility). *Let $\kappa = (N, \varphi = \langle F_0, F_1, \ldots, F_N \rangle, q)$ be a* PDR *configuration, and $\omega \in \Omega$. The intermediate forward sequence φ is* proof-compatible *with*

ω if $|\omega| = |\varphi| = N+1$ and for every $0 \leq i \leq N$, $F_i = \Sigma \setminus \omega(N-i)$. An obligation $(i, \sigma) \in q$ is cex-compatible *with ω if $|\omega| \geq i+1$ and $\sigma \in \omega(|\omega|-1-i)$.*

We say that κ is compatible *with a set of sequences $O \subseteq \Omega$, if*

1. *there exists $\omega_\varphi \in O$ such that φ is proof-compatible with ω_φ, and*
2. *either $q = \bot$ or for every obligation $\psi = (i, \sigma) \in q$, there exists $\omega_\psi \in O$ such that ψ is cex-compatible with ω_ψ.*

We refer to ω_φ and ω_ψ as the witnessing sequences *for φ and ψ, respectively.*

Thus, proof-compatibility requires that that sequences φ and ω are "mirrors" of each other combined with a pointwise complement operation. This also explains the choice of the term "anti-frames" for the sets in a backward cartesian trace. (See Section 5.) Cex-compatibility requires that the CTI σ which appears as an obligation in index i with respect to φ, will appear in ω in distance i from the *end* of the sequence.

Lemma 6. *The compatibility relation is a stuttering simulation between reachable* PDR *configurations and reachable elements of the collecting property-guided cartesian trace semantics.*

Proof. We prove the claim by showing that the initial configurations of ***PDR*** and the semantics are compatible, and that every step of ***PDR*** maintains compatibility.

Initial configuration. Let κ_0 be the initial configuration of ***PDR***, and $\hat{\Omega}$ be the initial element of the semantics. Then $\varphi_0 = \langle Init, \Sigma \setminus Bad \rangle$ is proof-compatible with the sequence $\hat{\omega}^2 = \langle Bad, \Sigma \setminus Init \rangle \in \hat{\Omega}$, and q is empty, hence cex-compatibility holds trivially.

Steps of* PDR.** Let $\kappa = (N, \varphi, q)$ be a configuration of ***PDR (where $N \geq 1$), and let O be an element of the semantics such that κ is compatible with O. For each possible step of ***PDR*** leading to $\kappa' = (N', \varphi', q')$, we show a corresponding sequence of $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$ leading from O to O' such that κ' is compatible with O'.

Note that it suffices to show sequences of transitions of $\mathrm{TR}^{BGE}_{\Omega}(P)$ leading to witnesses for φ' and for the obligations in q' separately. Since $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$ is monotonic and accumulative (i.e., if $\omega \in O$ and O has a transition of $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$ to O'', then $\omega \in O''$ as well), these sequences of transitions of $\mathrm{TR}^{BGE}_{\Omega}(P)$ can then be lifted to transitions of $\mathrm{TR}^{BGE}_{\mathscr{P}(\Omega)}(P)$, concatenated and applied on O to obtain O'. For the same reason it suffices to show such sequences of transitions only for the components in the ***PDR*** configuration that have changed in the step from κ to κ': for an unchanged component, the same witness from O, which exists in any subsequent element O'' of O, remains a witness.

Queue initialization: $\kappa' = (N, \varphi, q')$ where q' is either $\bot$, or a singleton $\{(N-1, \sigma)\}$. Consider first the case where $q' = \bot$. In this case, κ' is compatible with the same O, i.e. no transition of the semantics is needed.

Consider now the case where $q' = \{(N-1, \sigma)\}$, where $\sigma \in \overleftarrow{TR}(Bad) \cap F_{N-1}$. Recall that O is a reachable element of the semantics. Therefore, $\hat{\Omega} \subseteq O$. Starting from $\hat{\omega}^{N+1} \in \hat{\Omega} \subseteq O$ we apply a pre-transition of the semantics in index 0 of $\hat{\omega}^{N+1}$, adding the set $\{\sigma\}$ to $\hat{\omega}^{N+1}(1)$. The transition is applicable since $\sigma \in \overleftarrow{TR}(Bad)$ and $\hat{\omega}^{N+1}(0) = Bad$. The result is ω' of length $N+1$ such that $\sigma \in \omega'(1)$, where $1 = |\omega'|-1-(N-1)$. Hence $(N-1, \sigma)$ is cex-compatible with ω'.

Backward step: $\kappa' = (N, \varphi, q')$, where $q' = q \cup \{(i-1, \sigma)\}$. Let $\omega_{(i,\sigma')}$ be the witnessing sequence for the obligation (i, σ') which is the trigger for this step (where $|\omega_{(i,\sigma')}| \geq i+1$). Similarly to the case of queue initialization, we use a pre-transition of the semantics in index $|\omega_{(i,\sigma')}|-1-i$ of $\omega_{(i,\sigma')}$ to add $\{\sigma\}$ to $\omega_{(i,\sigma')}(|\omega_{(i,\sigma')}|-i)$, resulting in $\omega'_{(i,\sigma')}$ of the same length, such that $\sigma \in \omega'_{(i,\sigma')}(|\omega_{(i,\sigma')}|-i)$. Therefore, $\omega'_{(i,\sigma')}$ is a witness for cex-compatibility of the new obligation $(i-1, \sigma)$.

Obligation lifting: $q' = q \cup (\{i-1\} \times S)$. Similarly to the backward step, let $\omega_{(i,\sigma')}$ be the witnessing sequence for the obligation (i, σ') which is the trigger for the backward step responsible for lifting. A witness is obtained for all $(i-1, \sigma) \in \{i-1\} \times S$, by a pre-transition from $\omega_{(i,\sigma')}$ in index $|\omega_{(i,\sigma')}|-1-i$ adding S to $\omega_{(i,\sigma')}(|\omega_{(i,\sigma')}|-i)$. The pre-transition is applicable since $S \subseteq \overleftarrow{TR}(\sigma')$.

Blocking: $q' = q \setminus \{(i, \sigma')\}$, and $\varphi' = \langle F_0, \ldots, F_{i-1}, F_i \setminus \{\sigma'\}, F_{i+1}, \ldots, F_N \rangle$, where $1 \leq i \leq N$. Since q' is a subset of q, the same witnessing sequences for its obligations in O appear in every subsequent element of O. As for φ', let $\omega_\varphi \in O$ be a witnessing sequence for φ. Since $\overleftarrow{TR}(\sigma') \cap F_{i-1} = \emptyset$, we generate a witnessing sequence for φ' by applying a generalization transition on ω_φ at index $N-i$ (i.e., updating index $N-i$ based on $N-i+1$) using the set $Y = \{\sigma'\}$, similarly to the simulation of a generalization step of ***PDR*** (see below).

Generalization: In this case, $\varphi' = \langle F_0 \setminus S, \ldots, F_i \setminus S, F_{i+1}, \ldots, F_N \rangle$. Let $\omega_\varphi = \langle \Sigma \setminus F_N, \ldots, \Sigma \setminus F_0 \rangle$ be a witnessing sequence for φ in O. We obtain ω'_φ by a sequence of generalization transitions. For every $j = 1, \ldots, i$ (in increasing order), starting from $\omega^1 = \omega_\varphi$, we apply a generalization transition on $\omega^j = \langle \Sigma \setminus F_N, \ldots, \Sigma \setminus F_j, \Sigma \setminus (F_{j-1} \setminus S), \ldots, \Sigma \setminus (F_1 \setminus S), \Sigma \setminus F_0 \rangle$ in index $N-j$ (i.e., updating index $N-j$ based on $N-j+1$) using the set $Y = S$, leading to ω^{j+1}. By the requirements of Gen, $Init \cap S = \emptyset$ and $TR(F_{i-1}) \cap S = \emptyset$, i.e., $\overleftarrow{TR}(S) \subseteq \Sigma \setminus F_{i-1}$. Since $F_{j-1} \subseteq F_{i-1}$ for every $j \leq i$, we have that $\overleftarrow{TR}(S) \subseteq \Sigma \setminus F_{j-1}$. As such S indeed satisfies the requirements of a generalization transition in index $N-j$ of ω_j. Finally, ω^{i+1} is a witnessing sequence for φ'.

Inductive generalization: This step is similar to generalization, where now $\overleftarrow{TR}(S) \subseteq \Sigma \setminus F_{j-1}$ does not necessarily hold, but $\overleftarrow{TR}(S) \subseteq \Sigma \setminus (F_{j-1} \setminus S)$ holds (since $TR(F_{i-1} \setminus S) \cap S = \emptyset$). However, since the transitions are performed from $j = 1$ and up, when the generalization transition is performed on $\omega^j = \langle \Sigma \setminus F_N, \ldots, \Sigma \setminus F_j, \Sigma \setminus (F_{j-1} \setminus S), \ldots, \Sigma \setminus (F_1 \setminus S), \Sigma \setminus F_0 \rangle$ in index $N-j$ (i.e., updating index $N-j$ based on $N-j+1$) using the set $Y = S$, it is already the case that $\omega^j(N-j+i) = \Sigma \setminus (F_{j-1} \setminus S)$. Therefore, $\overleftarrow{TR}(S) \subseteq \omega^j(N-j+i)$ holds.

Forward propagation: $\varphi' = \langle F_0, \ldots, F_i, F_{i+1} \cup S, F_{i+2}, \ldots, F_N \rangle$. Let ω_φ be a witnessing sequence for φ in O. We obtain ω'_φ by a generalization transition on ω_φ in index $N-i-1$ (updating index $N-i-1$ based on $N-i$).

Pushing obligations forward: Recall that in this case $\kappa' = (N, \varphi, q \cup \{(i+1, \sigma)\})$. In this case, we show how to obtain a cex-witness ω' for $(i+1, \sigma)$ by a sequence of pre-transitions. By the property of the obligations in ***PDR***, there exists k and a sequence $\langle \sigma_k, \sigma_{k-1}, \ldots, \sigma_0 \rangle$ such that $\sigma_k = \sigma$ and $\sigma_0 \in Bad$ (i.e., σ leads to a bad state in k steps). Therefore, starting from $\omega^0 = \hat{\omega}^{i+2+k} \in \hat{\Omega} \subseteq O$ of length $i+2+k$,

we apply pre-transitions for every $j = 0, \ldots, k-1$ (in increasing order) in index j of ω^j, adding the singleton $\{\sigma_{j+1}\}$ to the $j+1$-th index, resulting in ω^{j+1} where $\omega^{j+1}(j+1) = \omega^j(j+1) \cup \{\sigma_{j+1}\}$. The result of the transitions is ω^k of length $i+2+k$ such that $\sigma \in \omega^k(k)$, where $k = |\omega^k| - 1 - (i+1)$. Hence $(i+1, \sigma)$ is cex-compatible with ω^k.

Unfolding: In this case, $\kappa' = (N+1, \langle F_0, F_1, \ldots, F_N, \Sigma \setminus Bad\rangle, \emptyset)$. We show how to obtain a witnessing sequence for $\varphi' = \langle F_0, F_1, \ldots, F_N, \Sigma \setminus Bad\rangle$ by a sequence of generalization transitions. We utilize again the property of reachable elements of the semantics which ensures that $\hat{\omega}^{N+2} = \langle Bad\rangle \emptyset^N \langle \Sigma \setminus Init\rangle \in \hat{\Omega} \subseteq O$. For every $i = 0, \ldots, N-1$ (in increasing order), starting from $\omega^0 = \hat{\omega}^{N+2}$, we apply a generalization transition on $\omega^i = \langle Bad\rangle \emptyset^{N-i} \langle \Sigma \setminus F_i, \ldots, \Sigma \setminus F_1, \Sigma \setminus Init\rangle$ in index $N-i$ (i.e., updating index $N-i$ based on index $N-i+1$) using the set $Y = \Sigma \setminus F_{i+1}$, leading to $\omega^{i+1} = \langle Bad\rangle \emptyset^{N-i-1} \langle \Sigma \setminus F_{i+1}, \ldots, \Sigma \setminus F_1, \Sigma \setminus Init\rangle$. To be convinced that the transition from ω^i to ω^{i+1} is well defined, we recall the properties of *PDR*. By the properties of *PDR*, for every $0 \leq i < N$, $TR(F_i) \subseteq F_{i+1}$, or equivalently, $\overleftarrow{TR}(\Sigma \setminus F_{i+1}) \subseteq \Sigma \setminus F_i$. In addition, $Init \subseteq F_{i+1}$, or equivalently $(\Sigma \setminus F_{i+1}) \cap Init = \emptyset$. As such, $Y = \Sigma \setminus F_{i+1}$ indeed satisfies the requirements of a generalization transition in index $N-i$ of ω^i. Finally, ω^N is a witnessing sequence for φ'. Since $q' = \emptyset$, no witnesses for cex-compatibility are needed. □

The proof of Lemma 6 shows that different components of the *PDR* configuration correspond to different sequences in the element of the semantics, O. In this sense, *PDR* can be thought of as trying to compute multiple sequences of the semantics simultaneously, as it both tries to find counterexamples of different lengths, and at the same time tries to verify safety.

Lemma 6 implies that all reachable configurations of *PDR* are compatible with reachable configurations of the semantics. This holds in particular for terminal configurations of *PDR*. We now show that the correctness of the output of *PDR* in each of the terminal configurations follows from their compatibility with an element of the semantics.

Counterexample: If there is an obligation $(0, \sigma') \in q$, *PDR* terminates and reports a counterexample. Such an obligation indicates that $\sigma' \in F_0$, i.e. $\sigma' \in Init$. Lemma 6 ensures that there is a reachable element O of the semantics with some $\omega \in O$ such that $\sigma' \in \omega(|\omega|-1)$. Indeed, since $\sigma' \in Init$, it follows that ω has an *error transition* leading to $\top$ (the error state of the semantics).

Fixpoint: If $q = \bot$, and there exists $i < N$ such that $F_i = F_{i+1}$, *PDR* terminates and reports safety. *PDR* has the property that when $q = \bot$, the intermediate forward sequence φ becomes a forward reachability sequence. Lemma 6 ensures that there is a reachable element O of the semantics with some $\omega \in O$ such that φ is proof-compatible with ω. Due to the properties of a forward reachability sequence (that hold for φ), and since $F_i \subseteq F_{i+1}$ and $TR(F_i) \subseteq F_{i+1}$ together imply $(\Sigma \setminus F_{i+1}) \cup \overleftarrow{TR}(\Sigma \setminus F_{i+1}) \subseteq \Sigma \setminus F_i$, it follows that ω has a projected fixpoint at its $N-i-1$ index.

Remark 2. *PDR* is sometimes implemented such that F_N is initialized to Σ rather than $\Sigma \setminus Bad$. In this case, in the intermediate forward sequences, requirement 4 of Def. 1 might not hold for $i = N$ (while requirement 3 holds for all frames). States that violate

requirement 4 are used as obligations at index N. Our semantics can simulate such implementations by letting a backward cartesian trace ω be a witness for an intermediate forward sequence φ if the *suffix* of ω in which the first anti-frame $\omega(0)$ is truncated is compatible with φ.

9 Discussion, Related Work and Conclusions

Implementations of *PDR* use a symbolic representation of states and sets of states, as formulas in logic. In the original description of *PDR* [3, 7], addressing finite state systems, propositional formulas over boolean variables are used. In this setting, which is most suitable for hardware designs, a SAT solver is used to preform one step reachability checks. In subsequent works which extended *PDR* to software, formulas in various theories of first order logic are considered, and SMT solvers are used instead of a boolean SAT solver. For example, [5] experiments with Linear Rational Arithmetic, [2, 9] handle Linear Real Arithmetic, [1] handles Linear Integer Arithmetic, and [10] considers universal formulas in first order logic. In our work, we use an explicit representation for the description of *PDR*, which captures all of these frameworks, in order to provide a view of *PDR* which is not restricted to a certain representation.

Our operational description of *PDR* is inspired by works such as [8,9] which provide an abstract description of *PDR* and its operations in the form of an abstract transition relation (described via formulas). However, we continue and show how this maps to a property-guided abstract interpretation of the program.

We consider linear *PDR*, where the semantics of a program is given via its traces (linear sequences). Some works (e.g. [5, 9]) have considered the extension of *PDR* to a non-linear search. [5] defined *tree-IC3* which can be thought of as performing *PDR* on each branch of a program's control flow graph. Handling such algorithms is the subject of future work.

Conclusions. We study, using abstract interpretation [6], the family of linear property directed reachability verification algorithms that has been developed following Bradley's original PDR/IC3 algorithm *PDR* [3]. We show that existing algorithms can be explained and proven sound by relating them to the actions of a non standard semantics which abstracts bounded backward traces. Arguably, the most surprising insight our work provides is that even though *PDR* is typically described as a forward analysis, it is in fact based on an abstraction of the *backward collecting trace* semantics. Besides the conceptual elegance of explaining existing algorithms (e.g. [1, 2, 7, 9, 10]) using (sequences of) two basic operations, we believe that our work would allow to explain and prove correct future *PDR*-based verification algorithms in a more systematic and abstract way than existing specialized techniques.

Acknowledgments We thank Mooly Sagiv, Eran Yahav, and the anonymous referees for their helpful comments. This work was supported by the European Research Council under the European Union's Seventh Framework Program (FP7/2007-2013) / ERC grant agreement no. [321174-VSSC], EU FP7 project ADVENT (308830), by Broadcom Foundation and Tel Aviv University Authentication Initiative, and by BSF grant no. 2012259.

References

1. J. Birgmeier, A. R. Bradley, and G. Weissenbacher. Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR). In *Computer Aided Verification, CAV*, pages 831–848, 2014.
2. N. Bjørner and A. Gurfinkel. Property directed polyhedral abstraction. In *Verification, Model Checking, and Abstract Interpretation - VMCAI*, pages 263–281, 2015.
3. A. Bradley. SAT-based model checking without unrolling. In *Verif., Model Checking, and Abs. Interp.*, 2011.
4. H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo. Incremental formal verification of hardware. In *International Conference on Formal Methods in Computer-Aided Design - FMCAD*, pages 135–143, 2011.
5. A. Cimatti and A. Griggio. Software model checking via IC3. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV'12, pages 277–293, 2012.
6. P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. In E. Neuhold, editor, *Formal Descriptions of Programming Concepts, (IFIP WG 2.2, St. Andrews, Canada, August 1977)*, pages 237–277. North-Holland, 1978.
7. N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, pages 125–134, 2011.
8. A. Gurfinkel. IC3, PDR and friends. arieg.bitbucket.org/pdf/gurfinkel_ssft15.pdf.
9. K. Hoder and N. Bjørner. Generalized property directed reachability. In *Theory and Applications of Satisfiability Testing - SAT*, pages 157–171, 2012.
10. A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetzky, and S. Shoham. Property-directed inference of universal invariants or proving their absence. In *Computer Aided Verification (CAV)*, 2015.