

Decidability and Synthesis of Abstract Inductive Invariants

Francesco Ranzato

Dipartimento di Matematica, University of Padova, Italy

Abstract. Decidability and synthesis of inductive invariants ranging in a given domain play an important role in many software and hardware verification systems. We consider here inductive invariants belonging to an abstract domain A as defined in abstract interpretation, namely, ensuring the existence of the best approximation in A of any system property. In this setting, we study the decidability of the existence of abstract inductive invariants in A of transition systems and their corresponding algorithmic synthesis. Our model relies on some general results which relate the existence of abstract inductive invariants with least fixed points of best correct approximations in A of the transfer functions of transition systems and their completeness properties. This approach allows us to derive decidability and synthesis results for abstract inductive invariants which are applied to the well-known Kildall’s constant propagation and Karr’s affine equalities abstract domains. Moreover, we show that a recent general algorithm for synthesizing inductive invariants in domains of logical formulae can be systematically derived from our results and generalized to a range of algorithms for computing abstract inductive invariants.

1 Introduction

Proof and inference methods based on inductive invariants are widespread in automatic (or semi-automatic) program and system verification (see, *e.g.*, [1, 3, 6, 9, 10, 11, 12, 18, 21, 22, 23, 26, 29, 32]). The inductive invariant proof method roots at the works of Floyd [14], Park [33, 34], Naur [31], Manna et al. [24] back to Turing [38]. Given a transition system $\mathcal{T} = \langle \Sigma, \tau, \Sigma_0 \rangle$, *e.g.* representing a program, where τ is a transition relation on states ranging in Σ and $\Sigma_0 \subseteq \Sigma$ is a set of initial states, together with a safety property $P \subseteq \Sigma$ to check, let us recall that a state property $I \subseteq \Sigma$ is an *inductive invariant* for $\langle \mathcal{T}, P \rangle$ when: $\Sigma_0 \subseteq I$, *i.e.* the initial states satisfy I ; $I \subseteq P$, *i.e.* I entails the safety property P ; $\tau(I) \subseteq I$, *i.e.* I is inductive. The *inductive invariant principle* states that P holds for all the reachable states of \mathcal{T} iff there exists an inductive invariant I for $\langle \mathcal{T}, P \rangle$. In this explicit form this principle has been probably first formulated by Cousot and Cousot [6, Section 5] in 1982 and called “induction principle for invariance proofs”. In most cases, verification and inference methods rely on inductive invariants I that range in some restricted domain $\mathcal{A} \subseteq \wp(\Sigma)$, such as a domain of logical formulae (*e.g.*, some separation logic or some fragment of first-order logic [32]) or a domain of abstract interpretation [4, 5] (*e.g.*, numerical abstract domains of affine relations or convex polyhedra). In this scenario, if an inductive invariant I belongs to \mathcal{A} then I is here called an *abstract inductive invariant*.

Main Contributions. Our primary goal was to investigate whether and how the inductive invariant principle can be adapted when inductive invariants are restricted to range in an abstract domain \mathcal{A} . We formulate this problem in general order-theoretical terms and we make the following working assumption: $\mathcal{A} \subseteq \wp(\Sigma)$ is an abstract domain as defined in abstract interpretation [4, 5]. This means that each state property $X \in \wp(\Sigma)$ has a *best* over-approximation (w.r.t. \subseteq) $\alpha_{\mathcal{A}}(X)$ in \mathcal{A} and each state transition relation τ has a *best* correct approximation $\tau^{\mathcal{A}}$ on the abstract domain \mathcal{A} . Under these hypotheses, we prove an *abstract inductive invariant principle* stating that there exists an abstract inductive invariant in \mathcal{A} proving a property P of a transition system \mathcal{T} iff the *best abstraction* $\mathcal{T}^{\mathcal{A}}$ in \mathcal{A} of the system \mathcal{T} allows us to prove P . The decidability/undecidability question of the existence of abstract inductive invariants in some abstract domain \mathcal{A} for some class of transition systems has been recently investigated in a few significant cases [13, 17, 28, 37]. We show how the abstract inductive

invariant principle allows us to derive a general decidability result on the existence of inductive invariants in some abstract domain \mathcal{A} and to design a general algorithm for synthesizing the least (w.r.t. the order of \mathcal{A}) abstract inductive invariant in \mathcal{A} , when this exists, by a least fixpoint computation in \mathcal{A} .

We also show a related result which is of independent interest in abstract interpretation: the (concrete) inductive invariant principle for a system \mathcal{T} is equivalent to the abstract inductive invariant principle for \mathcal{T} on an abstract domain \mathcal{A} iff *fixpoint completeness* of \mathcal{T} on \mathcal{A} holds, *i.e.*, the best abstraction in \mathcal{A} of the reachable states of \mathcal{T} coincides with the reachable states of the best abstraction $\mathcal{T}^{\mathcal{A}}$ of \mathcal{T} on \mathcal{A} .

The decidability/synthesis of abstract inductive invariants in a domain \mathcal{A} for some class \mathcal{C} of systems essentially boils down to prove that the best correct approximation $\tau^{\mathcal{A}}$ in \mathcal{A} of the transition relation τ of systems in the class \mathcal{C} is computable. As case studies, we provide two such results for Kildall's constant propagation [20] and Karr's affine equalities [19] domains, which are well-known and widely used abstract domains in numerical program analysis [25]. As a second application, we design an inductive invariant synthesis algorithm which, by generalizing a recent algorithm by Padon et al. [32] tailored for logical invariants, outputs the most abstract (*i.e.*, weakest) inductive invariant in a domain \mathcal{A} which satisfies some suitable hypotheses. In particular, we show that this synthesis algorithm is obtained by instantiating a concrete co-inductive (*i.e.*, based on a greatest fixpoint computation) fixpoint checking algorithm by Cousot [2] to a domain \mathcal{A} of abstract invariants which is *disjunctive*, *i.e.*, the abstract least upper bound of \mathcal{A} does not lose precision. This generalization allows us to design further related co-inductive algorithms for synthesizing abstract inductive invariants.

2 Background

2.1 Order Theory

If X is a subset of some universe set U then $\neg X$ denotes the complement of X with respect to U when U is implicitly given by the context. If $f : X \rightarrow Y$ is a function between sets and $S \in \wp(X)$ then $f(S) \triangleq \{f(x) \in Y \mid x \in S\}$ denotes the image of f on S . A composition of two functions f and g is denoted both by fg or $f \circ g$. If $\vec{x} \in X^n$ is a vector in a product domain, $j \in [1, n]$ and $y \in X$ then $\vec{x}[x_j/y]$ denotes the vector obtained from \vec{x} by replacing its j -th component x_j with y . To keep the notation simple and compact, we use the same symbol for a function/relation and its componentwise (*i.e.* pointwise) extension on product domains, *e.g.*, if $\vec{S}, \vec{T} \in \wp(X)^n$ then $\vec{S} \subseteq \vec{T}$ denotes that for all $i \in [1, n]$, $\vec{S}_i \subseteq \vec{T}_i$. Sometimes, to emphasize a pointwise definition, a dotted notation can be used such as in $f \dot{\leq} g$ for the pointwise ordering between functions.

A quasiordered set $\langle D, \leq \rangle$ (qoset), compactly denoted by D_{\leq} , is a set D endowed with a quasiorder (qo) relation \leq on D , *i.e.* a reflexive and transitive binary relation. A qoset D_{\leq} satisfies the ascending (resp. descending) chain condition (ACC, resp. DCC) if D contains no countably infinite sequence of distinct elements $\{x_i\}_{i \in \mathbb{N}}$ such that, for all $i \in \mathbb{N}$, $x_i \leq x_{i+1}$ (resp. $x_{i+1} \leq x_i$). An antichain in a qoset D is a subset $X \subseteq D$ such that any two distinct elements in X are incomparable for the qo \leq . A qoset D_{\leq} is a partially ordered set (poset) when \leq is antisymmetric. A subset $X \subseteq D$ of a poset is directed if X is nonempty and every pair of elements in X has an upper bound in X . A poset is a directed-complete partial order (CPO) if it has the least upper bound (lub) of all its directed subsets. A complete lattice is a poset having the lub of all its arbitrary (possibly empty) subsets (and therefore also having arbitrary glb's). In a complete lattice (or CPO), \vee (or \sqcup) and \wedge (or \sqcap) denote, resp., lub and glb, and \perp and \top denote, resp., least and greatest element.

Let P_{\leq} be a poset and $f : P \rightarrow P$. Then, $\text{Fix}(f) \triangleq \{x \in P \mid f(x) = x\}$, $\text{Fix}^{\leq}(f) \triangleq \{x \in P \mid f(x) \leq x\}$, $\text{Fix}^{\geq}(f) \triangleq \{x \in P \mid f(x) \geq x\}$, and $\text{lfp}(f)$ denote, resp., the least and greatest fixpoint in $\text{Fix}(f)$, when they exist. Let us recall Knaster-Tarski fixpoint theorem: if $\langle C, \leq, \vee, \wedge \rangle$ is a complete lattice and $f : C \rightarrow C$ is monotonic (*i.e.*, $x \leq y$ implies $f(x) \leq f(y)$) then $\langle \text{Fix}(f), \leq \rangle$ is a complete lattice, $\text{lfp}(f) = \bigwedge \text{Fix}^{\leq}(f)$ and $\text{gfp}(f) = \bigvee \text{Fix}^{\geq}(f)$. Also, Knaster-Tarski-Kleene fixpoint theorem states that if $\langle C, \leq, \vee, \perp \rangle$ is a CPO and $f : C \rightarrow C$ is Scott-continuous (*i.e.*, f preserves lub of directed subsets) then $\text{lfp}(f) = \bigvee_{i \in \mathbb{N}} f^i(\perp)$, where, for all $x \in C$ and $i \in \mathbb{N}$, $f^0(x) \triangleq x$ and $f^{i+1}(x) \triangleq f(f^i(x))$; dually, if $\langle C, \leq, \wedge, \top \rangle$ is a dual-CPO and $f : C \rightarrow C$

is Scott-co-continuous then $\text{gfp}(f) = \bigwedge_{i \in \mathbb{N}} f^i(\top)$. A function $f : C \rightarrow C$ on a complete lattice is additive when it preserves arbitrary lubs.

2.2 Abstract Domains

Let us recall some basic notions on closures and Galois connections which are commonly used in abstract interpretation [4, 5] to define abstract domains (see, e.g., [25, 36]). Closure operators and Galois connections are equivalent notions and are both used for defining the notion of approximation in abstract interpretation, where closure operators bring the advantage of defining abstract domains independently of a specific representation which is required by Galois connections.

An upper closure operator (uco), or simply upper closure, on a poset C_{\leq} is a function $\mu : C \rightarrow C$ which is: monotonic, idempotent and extensive, i.e., $x \leq \mu(x)$ for all $x \in C$. Dually, a lower closure operator (lco) $\eta : C \rightarrow C$ is monotonic, idempotent and reductive, i.e., $\eta(x) \leq x$ for all $x \in C$. The set of all upper/lower closures on C_{\leq} is denoted by $\text{uco}(C_{\leq})/\text{lco}(C_{\leq})$. We write $c \in \mu(C)$, or simply $c \in \mu$, to denote that there exists $c' \in C$ such that $c = \mu(c')$, and we recall that this happens iff $\mu(c) = c$. Let us also recall that $\langle \mu(C), \leq \rangle$ is closed under glb of arbitrary subsets and, conversely, $X \subseteq C$ is the image of some $\mu \in \text{uco}(C)$ iff X is closed under glb of all its subsets, and in this case $\mu(c) = \bigwedge \{c' \in X \mid c \leq c'\}$ holds. Dually, $X \subseteq C$ is closed under arbitrary lub of its subsets iff X is the image of a lower closure $\eta \in \text{lco}(C)$, and in this case $\eta(c) = \bigvee \{c' \in X \mid c' \leq c\}$. In abstract interpretation, a closure $\mu \in \text{uco}(C_{\leq})$ on a concrete domain C_{\leq} plays the role of an abstract domain having best approximations: $c \in C$ is approximated by any $\mu(c')$ such that $c \leq \mu(c')$ and $\mu(c)$ is the best approximation of c in μ because $\mu(c) = \bigwedge \{\mu(c') \mid c' \in C, c \leq \mu(c')\}$.

A Galois Connection (GC) (also called adjunction) between two posets $\langle C, \leq_C \rangle$, called concrete domain, and $\langle A, \leq_A \rangle$, called abstract domain, consists of two maps $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ such that $\alpha(c) \leq_A a \Leftrightarrow c \leq_C \gamma(a)$ holds. A GC is called Galois insertion (GI) when α is surjective or, equivalently, γ is injective. Any GC can be transformed into a GI simply by removing useless elements in $A \setminus \alpha(C)$ from the abstract domain A . A GC/GI is denoted by $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$. The function α is called the left-adjoint of γ , and, dually, γ is called the right-adjoint of α . This terminology is justified by the fact that if $\alpha : C \rightarrow A$ admits a right-adjoint $\gamma : A \rightarrow C$ then this is unique, and this dually holds for left-adjoints. GCs and ucos are equivalent notions because any GC $\mathcal{G} = (C, \alpha, \gamma, A)$ induces a closure $\mu_{\mathcal{G}} \triangleq \gamma \circ \alpha \in \text{uco}(C)$, any $\mu \in \text{uco}(C)$ induces a GI $\mathcal{G}_{\mu} \triangleq (C, \mu, \lambda x.x, \mu(C))$, and these two transforms are inverse of each other.

2.3 Transition Systems

Let $\mathcal{T} = \langle \Sigma, \tau \rangle$ be a transition system where Σ is a set of states and $\tau \subseteq \Sigma \times \Sigma$ is a transition relation. As usual, a transition relation can be equivalently defined by one of the following transformers of type $\wp(\Sigma) \rightarrow \wp(\Sigma)$:

$$\begin{aligned} \text{pre}(X) &\triangleq \{s \in \Sigma \mid \exists s' \in X. (s, s') \in \tau\} & \widetilde{\text{pre}}(X) &\triangleq \{s \in \Sigma \mid \forall s'. (s, s') \in \tau \Rightarrow s' \in X\} \\ \text{post}(X) &\triangleq \{s' \in \Sigma \mid \exists s \in X. (s, s') \in \tau\} & \widetilde{\text{post}}(X) &\triangleq \{s' \in \Sigma \mid \forall s. (s, s') \in \tau \Rightarrow s \in X\} \end{aligned}$$

We will equivalently specify a transition system by one of the above transformers (typically post) in place of the transition relation τ . Let us also recall (see e.g. [7]) that $\langle \text{pre}, \widetilde{\text{post}} \rangle$ and $\langle \text{post}, \widetilde{\text{pre}} \rangle$ are pairs of adjoint functions. The set of reachable states of \mathcal{T} from a set of initial states $\Sigma_0 \subseteq \Sigma$ is given by $\text{Reach}[\mathcal{T}, \Sigma_0] \triangleq \text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X))$, and \mathcal{T} satisfies a safety property $P \subseteq \Sigma$ when $\text{Reach}[\mathcal{T}, \Sigma_0] \subseteq P$ holds.

2.4 Inductive Invariant Principle

Given a transition system $\mathcal{T} = \langle \Sigma, \tau \rangle$, a set of states $I \in \wp(\Sigma)$ is an *inductive invariant* for \mathcal{T} w.r.t. $\langle \Sigma_0, P \rangle \in \wp(\Sigma)^2$ when: (i) $\Sigma_0 \subseteq I$; (ii) $\text{post}(I) \subseteq I$; (iii) $I \subseteq P$. An inductive invariant I allows us to prove that \mathcal{T} is safe,

i.e. $\text{Reach}[\mathcal{T}, \Sigma_0] \subseteq P$, by the *inductive invariant principle* (a.k.a. fixpoint induction principle), a consequence of Knaster-Tarski fixpoint theorem: If C_{\leq} is a complete lattice, $c' \in C$ and $f : C \rightarrow C$ is monotonic then

$$\text{lfp}(f) \leq c' \Leftrightarrow \exists i \in C. f(i) \leq i \wedge i \leq c' \quad (1)$$

In particular, given $c, c' \in C$, since $c \vee_C f(i) \leq i$ iff $c \leq i \wedge f(i) \leq i$, it turns out that:

$$\text{lfp}(\lambda x. c \vee_C f(x)) \leq c' \Leftrightarrow \exists i \in C. c \leq i \wedge f(i) \leq i \wedge i \leq c' \quad (2)$$

One such $i \in C$ such that $c \leq i \wedge f(i) \leq i \wedge i \leq c$ is called an *inductive invariant* of f for $\langle c, c' \rangle$. Hence, (2) is applied to the function $\lambda X. \Sigma_0 \cup \text{post}(X) : \wp(\Sigma) \rightarrow \wp(\Sigma)$, which is monotonic on the complete lattice $\wp(\Sigma)_{\subseteq}$, so that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X)) \subseteq P$ holds iff there exists an inductive invariant I for \mathcal{T} w.r.t. $\langle \Sigma_0, P \rangle$. In most interesting contexts for defining transition systems, the decision problem of the existence of a (concrete) inductive invariant for a class of transition systems w.r.t. a set of initial states and some safety property is undecidable.

3 Abstract Inductive Invariants

Padon et al. [32], Hrushovski et al. [17], Fijalkow et al. [13], Monniaux [28], Shoham [37], among the others, consider a notion of abstract inductive invariant and study the corresponding decidability/undecidability and synthesis problems. The common approach of this stream of works consists in restricting the range of inductive invariants from a concrete domain C to some abstraction A_C of C . In a basic and general form, a domain A_C of abstract invariants is simply a subset of C . Let us formalize abstract inductive invariants in general order-theoretic terms. Given a class \mathcal{C} of complete lattices and, for all $C \in \mathcal{C}$, a class of functions $\mathcal{F}_C \subseteq C \rightarrow C$, a set of initial properties $\text{Init}_C \subseteq C$, a set of safety properties $\text{Safe}_C \subseteq C$, and some abstract domain $A_C \subseteq C$, a first problem is the decidability of the following decision question:

$$\forall C \in \mathcal{C}. \forall f \in \mathcal{F}_C. \forall c \in \text{Init}_C. \forall c' \in \text{Safe}_C. \exists i \in A_C. c \leq i \wedge f(i) \leq i \wedge i \leq c' \quad (3)$$

where one such $i \in A_C$ is called an *abstract inductive invariant* for f and $\langle c, c' \rangle \in C^2$. On the other hand, the synthesis problem consists in designing algorithms which output abstract inductive invariants in A_C or notify that no inductive invariant in A_C exists.

Given a transition system $\mathcal{T} = \langle \Sigma, \tau \rangle$ whose successor transformer is post , the problem (3) is instantiated to $C_{\leq} = \wp(\Sigma)_{\subseteq}$, $f = \text{post}(X)$, $c = \Sigma_0 \in \wp(\Sigma)$ set of initial states and $c' = P \in \wp(\Sigma)$ safety property. When the transition system is generated by some imperative program, Σ_0 are the states of some initial control node and P is a safety property given by the states which are not in some bad control node, abstract inductive invariants are called *separating invariants* and the decision problem (3) is called *Monniaux problem* by Fijalkow et al. [13], because this was first formulated by Monniaux [27, 28].

3.1 Abstract Inductive Invariant Principle

Our working assumption is that in problem (3) invariants i range in an abstract domain A as defined in abstract interpretation [4, 5].

Assumption 3.1. $\langle A, \leq_A \rangle$ is an abstract domain of the complete lattice $\langle C, \leq_C \rangle$ which has best approximations, i.e., one of these two equivalent assumptions is satisfied:

- (i) $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$ is a Galois insertion;
- (ii) $\langle A, \leq_A \rangle = \langle \mu(C), \leq_C \rangle$ for some upper closure $\mu \in \text{uco}(C_{\leq_C})$. □

Under Assumption 3.1, let us recall that if $f : C \rightarrow C$ is a concrete monotonic function then the mappings $\alpha f \gamma : A \rightarrow A$, for the case of GIs, and $\mu f : \mu(C) \rightarrow \mu(C)$, for the case of ucos, are called *best correct approximation* (bca) in A of f . This is justified by the observation that an abstract function $f^\# : A \rightarrow A$ (or $f^\# : \mu(C) \rightarrow \mu(C)$ for ucos) is a correct (or sound) approximation of f when $\alpha f \gamma \dot{\leq}_A f^\#$ (or $\mu f \dot{\leq}_C f^\#$ for ucos) holds. Our first result is an *abstract inductive invariant principle* which restricts the invariants of f in (1) to those ranging in an abstract domain A : when the abstract domain A is specified by a GI, this means that $a \in A$ is an abstract invariant of f when $f\gamma(a) \leq_C \gamma(a)$ holds; when the abstract domain is a closure $\mu \in \text{uco}(C)$, this means that $a \in \mu \subseteq C$ is an abstract invariant of f when $fa \leq_C a$ holds.

Lemma 3.2 (Abstract Inductive Invariant Principle). *Let $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$ be a GI. For all $c' \in C$ and $a' \in A$:*

- (a) $\gamma(\text{lfp}(\alpha f \gamma)) \leq_C c' \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c'$;
- (b) $\text{lfp}(\alpha f \gamma) \leq_A a' \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a')$.

Proof. Let us first recall that in a GI, for all $a, a' \in A$, $a \leq_A a' \Leftrightarrow \gamma(a) \leq_C \gamma(a')$ holds.

(a) (\Leftarrow) We have that:

$$\begin{aligned}
\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c' &\Leftrightarrow \text{[by GC]} \\
\exists a \in A. \alpha f \gamma(a) \leq_A a \wedge \gamma(a) \leq_C c' &\Rightarrow \text{[by } Fx \leq x \Rightarrow \text{lfp}(F) \leq x\text{]} \\
\exists a \in A. \text{lfp}(\alpha f \gamma) \leq_A a \wedge \gamma(a) \leq_C c' &\Leftrightarrow \text{[by GI]} \\
\exists a \in A. \gamma(\text{lfp}(\alpha f \gamma)) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c' &\Rightarrow \text{[by transitivity]} \\
\gamma(\text{lfp}(\alpha f \gamma)) \leq_C c' &
\end{aligned}$$

(\Rightarrow) Define $a \triangleq \text{lfp}(\alpha f \gamma) \in A$. It turns out that $\alpha f \gamma(a) \leq_A a$ so that, by GC, $f\gamma(a) \leq_C \gamma(a)$, and, by hypothesis, $\gamma(a) \leq_C c'$.

(b) It turns out that:

$$\begin{aligned}
\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a') &\Leftrightarrow \text{[By Lemma 3.2 (a)]} \\
\gamma(\text{lfp}(\alpha f \gamma)) \leq_C \gamma(a') &\Leftrightarrow \text{[by GI]} \\
\text{lfp}(\alpha f \gamma) \leq_A a' &
\end{aligned}$$

□

It is worth stating Lemma 3.2 (a) in an equivalent form for an abstract domain represented by a closure $\mu \in \text{uco}(C)$: $\text{lfp}(\mu f) \leq_C c' \Leftrightarrow \exists a \in \mu. fa \leq_C a \wedge a \leq_C c'$.

Let us observe that point (b) is an easy consequence of point (a), because, by surjectivity of α in GIs, for all $a' \in A$, there exists some $c' \in C$ such that $a' = \alpha(c')$, and $\gamma(\text{lfp}(\alpha f \gamma)) \leq_C \gamma(\alpha(c')) \Leftrightarrow \text{lfp}(\alpha f \gamma) \leq_A \alpha(c')$ holds. Moreover, point (b) easily follows from the inductive invariant principle (1) for the bca $\alpha f \gamma : A \rightarrow A$. On the other hand, point (a) cannot be obtained from (b), i.e. (a) is *strictly stronger* than (b), because (a) allows us to prove concrete properties $c' \in C$ which are not exactly represented by A (i.e., $c' \notin \gamma(A)$) by abstract inductive invariants in A . This is shown by the following tiny example.

Example 3.3. Consider a 4-point chain $C = \{1 < 2 < 3 < 4\}$, the function $f : C \rightarrow C$ defined by $\{1 \mapsto 1; 2 \mapsto 2; 3 \mapsto 4; 4 \mapsto 4\}$, and the abstraction $A = \{2, 4\}$ with $\gamma = \text{id}$ and $\alpha = \{1 \mapsto 2; 2 \mapsto 2; 3 \mapsto 4; 4 \mapsto 4\}$. Here, we have that $\alpha f \gamma = \{2 \mapsto 2; 4 \mapsto 4\}$ and $\text{lfp}(\alpha f \gamma) = 2$. In this case, Lemma 3.2 (b) allows us to prove all the abstract properties $a' \in A$ by abstract inductive invariants, while Lemma 3.2 (a) allows us to prove an additional concrete property $3 \in C \setminus \gamma(A)$, which is not exactly represented by A , by an abstract inductive invariant, and this would not be possible by resorting to Lemma 3.2 (b). Also, $\gamma(\text{lfp}(\alpha f \gamma)) \not\leq 1$ holds, thus, by Lemma 3.2 (a), the concrete property 1 cannot be proved by an abstract inductive invariant in A , whereas Lemma 3.2 (b) does not allow us to infer this. □

Lemma 3.2 (b) tells us that the existence of an abstract inductive invariant of f proving an abstract property $a' \in A$ is equivalent to the fact that the least fixpoint of the best correct approximation $\alpha f \gamma$ entails a' . This formalizes for an abstract domain satisfying Assumption 3.1 an observation in [13, Section 1] stating that “the existence of some abstract inductive invariant for $\alpha f \gamma$ proving a' is equivalent to whether the strongest abstract invariant $\text{lfp}(\alpha f \gamma)$ entails a' ”, i.e. is inductive. If, instead, we aim at proving *any* concrete property $c' \in C$, possibly not in $\gamma(A)$, by an abstract inductive invariant then Lemma 3.2 (a) states that this is equivalent to the stronger condition $\gamma(\text{lfp}(\alpha f \gamma)) \leq_C c'$.

As a consequence of Lemma 3.2 (a) we derive the following characterization of the problem (3).

Corollary 3.4. *Let $\mathcal{F} \subseteq C \rightarrow C$ and $\text{Init}, \text{Safe} \subseteq C$. The Monniaux decision problem $\forall f \in \mathcal{F}. \forall c \in \text{Init}. \forall c' \in \text{Safe}. \exists a \in A. c \leq_C \gamma(a) \wedge f \gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c'$ is decidable iff the decision problem $\forall f \in \mathcal{F}. \forall c \in \text{Init}. \forall c' \in \text{Safe}. \gamma(\text{lfp}(\lambda x \in A. \alpha(c) \vee_A \alpha f \gamma(x))) \leq_C^? c'$ is decidable.*

Proof. By Lemma 3.2 (a), because $\lambda x \in A. \alpha(c) \vee_A \alpha f \gamma(x) = \lambda x \in A. \alpha(c \vee_C f \gamma(x))$ is the best correct approximation of $\lambda x \in C. c \vee_C f(x)$. \square

Moreover, as a consequence of Lemma 3.2 (b) we obtain the following abstract invariant synthesis algorithm.

Corollary 3.5. *Assume that the lub $\vee_A : A \times A \rightarrow A$ and the bca $\alpha f \gamma : A \rightarrow A$ are computable, the partial order $\leq_A^?$ is decidable and A is an ACC CPO. For all $c \in C$ such that $\alpha(c)$ is computable and $a' \in A$, the following procedure $\text{AINV}(f, A, c, a')$:*

```

i :=  $\alpha(c)$ ;
while  $i \leq_A a'$  do { if  $\alpha f \gamma(i) \leq_A i$  return  $i$ ; else  $i := \alpha f \gamma(i)$ ; }
return no abstract inductive invariant for  $f$  and  $\langle c, \gamma(a') \rangle$ ;

```

is a terminating algorithm which outputs the least abstract inductive invariant for f and $\langle c, \gamma(a') \rangle$, when one such abstract inductive invariant exists, otherwise outputs “no abstract inductive invariant”.

Proof. The hypotheses guarantee that the procedure AINV is a terminating algorithm, in particular because the sequence of computed iterates i is an ascending chain in A . If the algorithm AINV outputs i then $i = \text{lfp}(\lambda a. \alpha(c) \vee_A \alpha f \gamma(a)) \leq_A a'$, so that $i = \bigwedge \{a \in A \mid \alpha(c) \leq_A i, \alpha f \gamma(i) \leq_A i, i \leq_A a'\}$, that is, i is the least inductive invariant in A for f and $\langle c, \gamma(a') \rangle$. If the algorithm AINV outputs “no abstract inductive invariant for f and $\langle c, \gamma(a') \rangle$ ” then there exists $j \in \mathbb{N}$ such that $(\lambda a. \alpha(c) \vee_A \alpha f \gamma(a))^j(\perp_A) \not\leq_A a'$, so that $\text{lfp}(\lambda a. \alpha(c) \vee_A \alpha f \gamma(a)) \not\leq_A a'$, that is, there exists no inductive invariant in A for f and $\langle c, \gamma(a') \rangle$. \square

3.2 Fixpoint Completeness in Abstract Interpretation

Soundness in abstract interpretation (more in general, in static analysis) is a mandatory requirement stating that no false negative can occur, that is, abstract fixpoint computations correctly (over-)approximate the corresponding concrete semantics: if $f : C \rightarrow C$ and $f^\# : A \rightarrow A$ are the concrete and abstract monotonic transformers then *fixpoint soundness* means that $\alpha(\text{lfp}(f)) \leq_A \text{lfp}(f^\#)$ holds, so that a positive abstract check $\text{lfp}(f^\#) \leq_A a'$ entails that $\gamma(a')$ concretely holds, i.e., $\text{lfp}(f) \leq_C \gamma(a')$. Fixpoint soundness is usually obtained as a consequence of *pointwise soundness*: if $f^\#$ is a pointwise correct approximation of f , i.e. $\alpha f \dot{\leq}_A f^\# \alpha$, then $\alpha(\text{lfp}(f)) \leq_A \text{lfp}(f^\#)$ holds.

While soundness is indispensable, completeness in abstract interpretation encodes an ideal situation where no false positives (also called false alarms) arise: *fixpoint completeness* means that $\alpha(\text{lfp}(f)) = \text{lfp}(f^\#)$ holds, so that $\text{lfp}(f^\#) \not\leq_A a'$ entails $\text{lfp}(f) \not\leq_C \gamma(a')$. One can also consider a *strong fixpoint completeness* requiring that $\text{lfp}(f) = \gamma(\text{lfp}(f^\#))$, so that $\text{lfp}(f^\#) \not\leq_A \alpha(c')$ entails $\text{lfp}(f) \not\leq_C c'$. However, it should be remarked that $\text{lfp}(f) = \gamma(\text{lfp}(f^\#))$ is much stronger than $\alpha(\text{lfp}(f)) = \text{lfp}(f^\#)$ since it means that the concrete lfp is precisely represented by the abstract lfp .

It is important to remark that if f^\sharp is a pointwise correct approximation of f and fixpoint completeness for f^\sharp holds then since $\alpha(\text{lfp}(f)) \leq_A \text{lfp}(\alpha f \gamma) \leq_A \text{lfp}(f^\sharp)$ always holds, one also obtains that $\alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma) = \text{lfp}(f^\sharp)$ holds, namely, the best correct approximation $\alpha f \gamma$ is fixpoint complete as well. This means that the possibility (or impossibility) of defining an approximate transformer $f^\sharp : A \rightarrow A$ on A which is fixpoint complete does not depend on the specific definition of f^\sharp but is instead an intrinsic *property of the abstract domain* A w.r.t. the concrete transformer f , as formalized by the equation $\alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma)$. Moreover, fixpoint completeness is typically proved as a by-product of *pointwise completeness* $\alpha f = f^\sharp \alpha$, and if f^\sharp is pointwise complete then it turns out that $f^\sharp = \alpha f \gamma$, that is, f^\sharp actually is the bca of f . This justifies why, without loss of generality, we can consider fixpoint and pointwise completeness of bca's $\alpha f \gamma$ only, *i.e.*, a property of abstract domains.

3.3 Characterizing Fixpoint Completeness by Abstract Inductive Invariants

We show that the abstract inductive invariant principle is closely related to fixpoint completeness. More precisely, we provide an answer to the following question: in the abstract inductive invariant principle as stated by Lemma 3.2, can we replace $\text{lfp}(\alpha f \gamma)$ with $\alpha(\text{lfp}(f))$? This question is settled by the following result.

Theorem 3.6. *Let $(C \leq_C, \alpha, \gamma, A \leq_A)$ be a GI.*

- (a) $\text{lfp}(f) = \gamma(\text{lfp}(\alpha f \gamma))$ iff $\forall c' \in C. (\text{lfp}(f) \leq_C c' \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c')$;
- (b) $\alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma)$ iff $\forall a' \in A. (\text{lfp}(f) \leq_C \gamma(a') \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a'))$.

Proof. (a) (\Rightarrow) : By Lemma 3.2 (a).

(\Leftarrow) : Since $\text{lfp}(f) \leq_C \text{lfp}(f)$ holds, we have that $\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \text{lfp}(f)$. Thus, by Lemma 3.2 (a), $\gamma(\text{lfp}(\alpha f \gamma)) \leq_C \text{lfp}(f)$ follows. On the other hand, $\text{lfp}(f) \leq \gamma(\text{lfp}(\alpha f \gamma))$ always holds because the pointwise correctness of $\alpha f \gamma$ implies $\alpha(\text{lfp}(f)) \leq_A \text{lfp}(\alpha f \gamma)$, hence, by GC, $\text{lfp}(f) \leq \gamma(\text{lfp}(\alpha f \gamma))$ follows.

(b) (\Rightarrow) : By Lemma 3.2 (b) because $\text{lfp}(\alpha f \gamma) \leq_A a' \Leftrightarrow \alpha(\text{lfp}(f)) \leq a' \Leftrightarrow \text{lfp}(f) \leq_C \gamma(a')$.

(\Leftarrow) : We consider $a' \triangleq \alpha(\text{lfp}(f))$, so that, $\text{lfp}(f) \leq_C \gamma(a')$ holds and by the equivalence of the hypothesis, $\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\text{lfp}(f))$ holds. This implies, by GI, that $\exists a \in A. \alpha f \gamma(a) \leq_C a \wedge a \leq_A \alpha(\text{lfp}(f))$. By the inductive invariant principle (1), this implies that (actually, is equivalent to) $\text{lfp}(\alpha f \gamma) \leq \alpha(\text{lfp}(f))$. Furthermore, $\alpha(\text{lfp}(f)) \leq_A \text{lfp}(\alpha f \gamma)$ always holds, therefore proving that $\alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma)$. \square

Theorem 3.6 (b) can be stated by means of ucos as follows: if $\mu \in \text{uco}(C)$ then $\mu(\text{lfp}(f)) = \text{lfp}(\mu f)$ iff $\forall a' \in \mu. (\text{lfp}(f) \leq_C a' \Leftrightarrow \exists a \in \mu. fa \leq_C a \wedge a \leq_C a')$.

The above result can be read as follows. Since, by the inductive invariant principle (1), $\text{lfp}(f) \leq_C c'$ iff there exists a (concrete) inductive invariant proving c' , it turns out that Theorem 3.6 (a) states that, for all $c' \in C$, the existence of an *abstract* inductive invariant proving c' is equivalent to the existence of *any* inductive invariant proving c' iff fixpoint completeness holds. In other terms, the (concrete) inductive invariant principle is equivalent to the abstract inductive invariant principle iff fixpoint completeness holds. This result is of independent interest in abstract interpretation, since it provides a new characterization of the key property of fixpoint completeness of abstract domains [15].

A further interesting characterization of fixpoint completeness is as follows.

Lemma 3.7. $\alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma) \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\text{lfp}(f))$.

Proof.

$$\begin{aligned}
\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\text{lfp}(f)) &\Leftrightarrow \text{[by GI]} \\
\exists a \in A. \alpha f \gamma(a) \leq_A a \wedge a \leq_A \alpha(\text{lfp}(f)) &\Leftrightarrow \text{[by (1) for } \alpha f \gamma\text{]} \\
\text{lfp}(\alpha f \gamma) \leq_A \alpha(\text{lfp}(f)) &\Leftrightarrow \text{[as } \alpha(\text{lfp}(f)) \leq_A \text{lfp}(\alpha f \gamma)\text{]} \\
\alpha(\text{lfp}(f)) &= \text{lfp}(\alpha f \gamma)
\end{aligned}$$

\square

As a consequence, fixpoint completeness for f does not hold in A iff the abstract property $\alpha(\text{lfp}(f)) \in A$ cannot be proved by an abstract inductive invariant in A

Example 3.8. Consider the finite chain $C \triangleq \{1 < 2 < 3\}$ and the monotonic concrete function $f : C \rightarrow C$ defined by $f \triangleq \{1 \mapsto 1; 2 \mapsto 3; 3 \mapsto 3\}$. Consider the uco $\mu \triangleq \{2, 3\}$, i.e., $\mu = \{1 \mapsto 2; 2 \mapsto 2; 3 \mapsto 3\}$. Hence, $\mu f = \{1 \mapsto 2; 2 \mapsto 3; 3 \mapsto 3\}$ so that fixpoint completeness does not hold because $\mu(\text{lfp}(f)) = \mu(1) = 2 < 3 = \text{lfp}(\mu f)$. Thus, in accordance with Lemma 3.7, it turns out that $\mu(\text{lfp}(f)) = 2$ cannot be inductively proved in the abstraction μ . In fact, $f(2) \not\leq 2$, while $f(3) \leq 3$ but $3 \not\leq \mu(\text{lfp}(f))$. Consider instead the uco $\mu \triangleq \{1, 3\}$, i.e., $\mu = \{1 \mapsto 1; 2 \mapsto 3; 3 \mapsto 3\}$, so that $\mu f = \{1 \mapsto 1; 2 \mapsto 3; 3 \mapsto 3\}$. Here, $\mu(\text{lfp}(f)) = \mu(1) = 1 = \text{lfp}(\mu f)$, therefore fixpoint completeness holds. Thus, by the uco version of Theorem 3.6 (b), any valid abstract invariant of f can be inductively proved: in fact, $1, 3 \in \mu$ are valid abstract invariants of f and are both inductive. \square

3.4 When Safety = Abstract Invariance?

Padon et al. [32, Section 9] in their investigation on the decidability of inferring inductive invariants state that “Usually completeness for abstract interpretation means that the abstract domain is precise enough to prove all interesting safety properties, e.g., [15]. In our terms, this means that $\text{SAFE} = \text{INV}$, that is, that all safe programs have an inductive invariant expressible in the abstract domain.” As a by-product of the results in Section 3.3, we are able to give a formal justification and statement of this informal characterization of completeness.

Let $\mathcal{F} \subseteq C \rightarrow C$ be a class of monotonic functions, $\mathcal{S} \subseteq C$ be some set of safety properties and $\mathcal{A} \subseteq C$ be an abstract domain of program properties. Let us define:

$$\begin{aligned} \text{SAFE}[\mathcal{F}, \mathcal{S}] &\triangleq \{\langle f, s \rangle \in \mathcal{F} \times \mathcal{S} \mid \text{lfp}(f) \leq_C s\} \\ \text{INV}[\mathcal{F}, \mathcal{S}, \mathcal{A}] &\triangleq \{\langle f, s \rangle \in \mathcal{F} \times \mathcal{S} \mid \exists a \in \mathcal{A}. fa \leq_C a \wedge a \leq_C s\} \end{aligned}$$

so that in our model $\text{SAFE}[\mathcal{F}, \mathcal{S}]$ and $\text{INV}[\mathcal{F}, \mathcal{S}, \mathcal{A}]$ play the role of, resp., “safe programs” and “programs having an inductive invariant expressible in \mathcal{A} ”. As a consequence of Theorem 3.6, we derive the following characterization.

Corollary 3.9. Assume that \mathcal{A} satisfies Assumption 3.1 for some $GI \langle C, \alpha, \gamma, \mathcal{A} \rangle$.

- (a) Assume that $\mathcal{S} \subseteq \mathcal{A}$. Then, $\text{SAFE}[\mathcal{F}, \mathcal{S}] = \text{INV}[\mathcal{F}, \mathcal{S}, \mathcal{A}]$ iff $\forall f \in \mathcal{F}. \alpha(\text{lfp}(f)) = \text{lfp}(\alpha f \gamma)$.
- (b) $\text{SAFE}[\mathcal{F}, \mathcal{S}] = \text{INV}[\mathcal{F}, \mathcal{S}, \mathcal{A}]$ iff $\forall f \in \mathcal{F}. \text{lfp}(f) = \gamma(\text{lfp}(\alpha f \gamma))$.

Proof. (a) follows by Theorem 3.6 (a), since $\mathcal{S} \subseteq \mathcal{A}$ is assumed to hold. (b) follows by Theorem 3.6 (b). \square

Corollary 3.9 therefore provides a precise equivalence of the $\text{SAFE} \stackrel{?}{=} \text{INV}$ problem, as stated by Padon et al. [32], with fixpoint completeness (strong fixpoint completeness, in case (b)) in abstract interpretation.

4 Abstract Inductive Invariants of Programs

We consider transition systems as represented by a finite control flow graph (CFG) of an imperative program. A program is a tuple $\mathcal{P} = \langle Q, n, \mathbb{V}, \mathbb{T}, \rightarrow \rangle$ where Q is a finite set of control nodes (or program points), $n \in \mathbb{N}$ is the number of program variables of type \mathbb{V} (e.g., $\mathbb{V} = \mathbb{Z}, \mathbb{Q}, \mathbb{R}$), \mathbb{T} is a finite set of (possibly nondeterministic) transfer functions of type $\mathbb{V}^n \rightarrow \wp(\mathbb{V}^n)$, $\rightarrow \subseteq Q \times \mathbb{T} \times Q$ is a (possibly nondeterministic) control flow relation, where $q \xrightarrow{t} q'$ denotes a flow transition with transfer function $t \in \mathbb{T}$. A program \mathcal{P} therefore defines a transition system $\mathcal{T}_{\mathcal{P}} = \langle \Sigma, \tau \rangle$ where $\Sigma \triangleq Q \times \mathbb{V}^n$ is the set of states and the transition relation $\tau \subseteq \Sigma \times \Sigma$ is defined by

$\langle (q, \vec{v}), (q', \vec{v}') \rangle \in \tau \hat{\Leftrightarrow} \exists t \in \mathbf{T}. q \xrightarrow{t} q' \wedge \vec{v}' \in t(\vec{v})$. The transfer functions in \mathbf{T} include imperative assignments and Boolean guards: if $b \in \wp(\mathbb{V}^n)$ is a deterministic Boolean predicate (such as $x_1 + 2x_2 - 1 = 0$) then the corresponding transfer function $t_b : \mathbb{V}^n \rightarrow \wp(\mathbb{V}^n)$ is $t_b(\vec{v}) \triangleq \text{if } \vec{v} \in b \text{ then } \{\vec{v}\} \text{ else } \emptyset$. Examples of transfer functions include: affine, polynomial, nondeterministic assignments and affine equalities guards.

The next value transformer $\text{post}_{\langle q, q' \rangle} : \wp(\mathbb{V}^n) \rightarrow \wp(\mathbb{V}^n)$ for a pair $\langle q, q' \rangle \in Q \times Q$ of control nodes is $\text{post}_{\langle q, q' \rangle}(X) \triangleq \cup \{t(X) \in \wp(\mathbb{V}^n) \mid \exists t \in \mathbf{T}. q \xrightarrow{t} q'\}$. Hence, if, for all $t \in \mathbf{T}$, $q \not\xrightarrow{t} q'$ then $\text{post}_{\langle q, q' \rangle} = \lambda X. \emptyset$. The complete lattice $\langle \wp(\Sigma), \subseteq \rangle$ of sets of states can be equivalently represented by the Q -indexed product lattice $\langle \wp(\mathbb{V}^n)^{|Q|}, \subseteq \rangle$. Hence, the successor transformer $\text{post}_{\mathcal{P}} : \wp(\mathbb{V}^n)^{|Q|} \rightarrow \wp(\mathbb{V}^n)^{|Q|}$ and the set of reachable states of a program \mathcal{P} from initial states in $\Sigma_0 \in \wp(\mathbb{V}^n)^{|Q|}$ are defined as follows:

$$\begin{aligned} \text{post}_{\mathcal{P}}(\langle X_q \rangle_{q \in Q}) &\triangleq \langle \cup_{q \in Q} \text{post}_{\langle q, q' \rangle}(X_q) \rangle_{q' \in Q} \\ \text{Reach}[\mathcal{P}, \Sigma_0] &\triangleq \text{lfp}(\lambda \vec{X}. \Sigma_0 \cup \text{post}_{\mathcal{P}}(\vec{X})) \in \wp(\mathbb{V}^n)^{|Q|} \end{aligned}$$

For all control nodes $q \in Q$ and vector $\vec{X} \in \wp(\mathbb{V}^n)^{|Q|}$, we will also use $\pi_q(\vec{X})$ and \vec{X}_q to denote the q -indexed component of \vec{X} , *e.g.*, $\text{Reach}[\mathcal{P}, \Sigma_0]_q \in \wp(\mathbb{V}^n)$ will be the set of reachable values at control node q .

We are interested in decidability and synthesis of abstract inductive invariants ranging in an abstract domain A as specified by a GI $(\wp(\mathbb{V}^n)_{\subseteq}, \alpha, \gamma, A_{\leq_A})$ parametric on $n \in \mathbb{N}$. By Corollary 3.4, for a given class \mathcal{C} of programs, a class Init of sets of initial states and a class Safe of sets of safety properties, the Monniaux problem (3) is decidable iff for all $\mathcal{P} = \langle Q, n, \mathbb{V}, \mathbf{T}, \rightarrow \rangle \in \mathcal{C}$, $\Sigma_0 \in \text{Init}$ and $P \in \text{Safe}$,

$$\dot{\gamma}(\text{lfp}(\lambda \vec{a} \in A^{|Q|}. \dot{\alpha}(\Sigma_0) \dot{\vee}_A \dot{\alpha}(\text{post}_{\mathcal{P}}(\dot{\gamma}(\vec{a})))))) \stackrel{?}{\subseteq} P \quad (4)$$

is decidable. Moreover, Corollary 3.5 provides an abstract inductive invariant synthesis algorithm AINV for safety properties represented by A (*i.e.*, $P \in \gamma(A)$) when A , \mathcal{C} , Init and Safe satisfy the hypotheses of Corollary 3.5.

In the following, we consider the decidability/synthesis problem (3) for Kildall's constant propagation [20] and Karr's affine equalities [19] abstract domains.

4.1 Kildall's Constant Propagation Domain

Kildall's constant propagation [20] is a well-known and simple program analysis widely used in compiler optimization for detecting whether a variable at some program point always stores a constant value for all possible program executions. Constant propagation relies on the abstract domain CONST , which, for simplicity, is here given for program variables assuming integer values in \mathbb{Z} (\mathbb{Q} and \mathbb{R} would be analogous). For a single variable $\text{CONST} \triangleq \langle \mathbb{Z} \cup \{\perp, \top\}, \leq \rangle$, where \leq is the standard flat partial order: for any $a \in \mathbb{Z} \cup \{\perp, \top\}$, $\perp \leq a \leq \top$ (and $a \leq a$), which makes CONST an infinite complete lattice with height 2. CONST is straightforwardly specified by a GI $(\wp(\mathbb{Z})_{\subseteq}, \alpha_C, \gamma_C, \text{CONST}_{\leq})$ where:

$$\alpha_C(X) \triangleq \begin{cases} \perp & \text{if } X = \emptyset \\ z & \text{if } X = \{z\} \\ \top & \text{otherwise} \end{cases} \quad \gamma_C(a) \triangleq \begin{cases} \emptyset & \text{if } a = \perp \\ \{a\} & \text{if } a \in \mathbb{Z} \\ \mathbb{Z} & \text{if } a = \top \end{cases}$$

For $n \geq 1$ variables, the product constant domain is $\text{CONST}_n \triangleq (\text{CONST}_{\perp})^n \cup \{\perp\}$, where $\text{CONST}_{\perp} \triangleq \text{CONST} \setminus \{\perp\}$, so as to have a unique bottom element representing the empty set, while $\alpha_{\text{CONST}} : \wp(\mathbb{Z}^n) \rightarrow \text{CONST}$ and $\gamma_{\text{CONST}} : \text{CONST} \rightarrow \wp(\mathbb{Z}^n)$ are defined as pointwise lifting of, resp., α_C and γ_C . $\langle \text{CONST}_n, \leq \rangle$ is a complete lattice of finite height $2n$, where lub and glb are defined pointwise. Finally, for a finite set of control nodes Q , $\dot{\alpha}_{\text{CONST}} : \wp(\mathbb{Z}^n)^{|Q|} \rightarrow \text{CONST}^{|Q|}$ and $\dot{\gamma}_{\text{CONST}} : \text{CONST}^{|Q|} \rightarrow \wp(\mathbb{Z}^n)^{|Q|}$ are the Q -indexed pointwise lifting of, resp., α_{CONST} and γ_{CONST} .

It is known since [16,35] that the constant propagation problem is undecidable, meaning that for any program \mathcal{P} with n variables, set Σ_0 of initial states, node q of \mathcal{P} , vector of constants (or \top) $\vec{k} \in \text{CONST}_n \setminus \{\perp\}$, the problem $\alpha_{\text{CONST}}(\text{Reach}[\mathcal{P}, \Sigma_0]_q) \leq^? \vec{k}$ is undecidable. This undecidability is obtained by a simple reduction from the undecidable Post correspondence problem and holds even if the interpretation of program branches is neglected, that is, in CFGs with unguarded nondeterministic branching.

By Corollary 3.4, the existence of abstract inductive invariants in CONST for a given class \mathcal{C} of programs, Init of sets of initial states and Safe of sets of safety properties, is a decidable problem iff for all $\mathcal{P} \in \mathcal{C}$ with n variables and control nodes in Q , for all $\Sigma_0 \in \text{Init}_{\mathcal{P}} \subseteq \wp(\mathbb{V}^n)^{|Q|}$ and for all $P \in \text{Safe}_{\mathcal{P}} \subseteq \wp(\mathbb{V}^n)^{|Q|}$,

$$\dot{\gamma}_{\text{CONST}}(\text{lfp}(\lambda \vec{a} \in \text{CONST}_n^{|Q|}. \dot{\alpha}_{\text{CONST}}(\Sigma_0) \dot{\vee}_{\text{CONST}_n} \dot{\alpha}_{\text{CONST}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(\vec{a})))) \dot{\subseteq}^? P \quad (5)$$

is decidable. We observe that:

- (i) if P is a recursive set and $\vec{a} \in \text{CONST}_n^{|Q|}$ then $\dot{\gamma}_{\text{CONST}}(\vec{a}) \dot{\subseteq}^? P$ is clearly decidable, because for all $q \in Q$, $i \in [1, n]$ and $k \in \mathbb{Z}$, $k \in^? \pi_i(\pi_q(P))$ is decidable (and any $\dot{\gamma}_{\text{CONST}}(\vec{a}')$ is trivially recursive);
- (ii) if Σ_0 is a recursively enumerable (r.e.) set then $\dot{\alpha}_{\text{CONST}}(\Sigma_0)$ is clearly computable, because since Σ_0 is r.e., for all $q \in Q$ and $i \in [1, n]$, a (double) projection $\pi_i(\pi_q(\Sigma_0))$ is r.e., so that an algorithm enumerating $\pi_i(\pi_q(\Sigma_0))$ allows us to compute its abstraction α_{C} in CONST ;
- (iii) the binary lub $\vec{a} \vee_{\text{CONST}_n} \vec{a}'$ in CONST_n is clearly (pointwise) computable, thus, in turn, the Q -indexed lub $\dot{\vee}_{\text{CONST}_n}$ is computable.

Therefore, since $\text{CONST}_n^{|Q|}$ has finite height, a sufficient condition for the decidability of the problem (5) is that the best correct approximation $\dot{\alpha}_{\text{CONST}} \circ \text{post}_{\mathcal{P}} \circ \dot{\gamma}_{\text{CONST}} : \text{CONST}_n^{|Q|} \rightarrow \text{CONST}_n^{|Q|}$ is computable, where for all $\vec{a} \in \text{CONST}_n^{|Q|}$:

$$\begin{aligned} & \dot{\alpha}_{\text{CONST}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(\vec{a}))) = \\ & \langle \vee_{\text{CONST}} \{ \alpha_{\text{CONST}}(\text{post}_{q,q'}(\pi_q(\dot{\gamma}_{\text{CONST}}(\vec{a})))) \mid q \in Q \} \rangle_{q' \in Q} = \\ & \langle \vee_{\text{CONST}} \{ \alpha_{\text{CONST}}(t(\pi_q(\dot{\gamma}_{\text{CONST}}(\vec{a})))) \mid \exists q \in Q, \exists t \in \mathbb{T}. q \xrightarrow{t} q' \} \rangle_{q' \in Q} \end{aligned} \quad (6)$$

Because in (6) we have a finite lub, it is enough that for all the transfer functions $t \in \mathbb{T}$ and $a \in \text{CONST}_n$, the bca $\alpha_{\text{CONST}}(t(\gamma_{\text{CONST}}(a)))$ is computable. It turns out that the best correct approximations in CONST of affine assignments and linear Boolean guards are computable. Some simple cases are described, e.g., in [25, Section 4.3], in the following we provide the algorithm of the bca's in full generality and detail.

Let $\sum_{i=1}^n m_i x_i + b \in \text{LEXP}_n$ be a linear expression for n integer variables ranging in $\text{Var}_n \triangleq \{x_1, \dots, x_n\}$, where $\langle m_i \rangle_{i=1}^n \in \mathbb{Z}^n$ is a vector of integer coefficients and $b \in \mathbb{Z}$, and let $\llbracket \sum_{i=1}^n m_i x_i + b \rrbracket : \wp(\mathbb{Z}^n) \rightarrow \wp(\mathbb{Z})$ denote its collecting semantics, that is, $\llbracket \sum_{i=1}^n m_i x_i + b \rrbracket X \triangleq \{ \sum_{i=1}^n m_i v_i + b \in \mathbb{Z} \mid \vec{v} \in X \}$. Then, we consider the transfer functions given by a single affine assignment $t_a^j \triangleq x_j := \sum_{i=1}^n m_i x_i + b$ and a linear Boolean guard $t_b \triangleq \sum_{i=1}^n m_i x_i + b \bowtie 0$ with $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$, where for all $Y \in \wp(\mathbb{Z}^n)$:

$$\begin{aligned} t_a^j(Y) & \triangleq \{ \vec{y}[y_j/v_j] \in \mathbb{Z}^n \mid \vec{y} \in Y, v_j \in \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket \{ \vec{y} \} \}, \\ t_b(Y) & \triangleq \{ \vec{y} \in Y \mid \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket \{ \vec{y} \} \bowtie 0 \}. \end{aligned}$$

These transfer functions can be easily extended to include parallel affine assignments of the shape $\vec{x} := M\vec{x} + \vec{b}$, where $M \in \mathbb{Z}^{n \times n}$ is a $n \times n$ integer matrix and $\vec{b} \in \mathbb{Z}^n$, which simply performs n parallel single affine assignments, and conjunctive (disjunctive) linear Boolean guards $M\vec{x} + \vec{b} \bowtie 0$, which holds iff, for all (there exists) $j \in [1, n]$, $\sum_{i=1}^n M_{ji} x_i + b_j \bowtie 0$ holds.

It turns out that the bca of t_a^j and t_b in CONST_n are both computable. Let us first observe that if $\vec{a} \in \text{CONST}_n \setminus \{\perp\}$ then $\llbracket \sum_{i=1}^n m_i x_i + b \rrbracket (\gamma_{\text{CONST}}(\vec{a})) = \{k\}$ for some $k \in \mathbb{Z}$ iff for all $l \in [1, n]$ either $m_l = 0$

or $a_i \in \mathbb{Z}$. Thus, for all $a \in \text{CONST}_n$, the problem $\exists k \in ? \mathbb{Z}. \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(a)) = \{k\}$ is decidable, and in a positive case the constant k is computable. Moreover, if $\llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) \neq \{k\}$ then necessarily $\llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \mathbb{Z}$. As a consequence of these observations, we derive the following definition: for all $a \in \text{CONST}_n$,

$$\alpha_{\text{CONST}}(t_a^j(\gamma_{\text{CONST}}(a))) = \begin{cases} \perp & \text{if } a = \perp \\ \vec{a}[a_j/k] & \text{if } a = \vec{a}, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \{k\} \\ \vec{a}[a_j/\top] & \text{if } a = \vec{a}, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \mathbb{Z} \end{cases}$$

which shows that $\alpha_{\text{CONST}}(t_a^j(\gamma_{\text{CONST}}(a)))$ is computable and, in turn, bca's of parallel affine assignments are computable. The proof of computability of bca's of linear Boolean guards t_b follows the same lines. We distinguish two cases: $\bowtie \in \{\neq, <, \leq, >, \geq\}$, whose transfer function is denoted by t_b^{\bowtie} , and $\bowtie \in \{=\}$, i.e. affine equalities, denoted by t_b^- . The corresponding algorithms are as follows: for all $a \in \text{CONST}_n$:

$$\alpha_{\text{CONST}}(t_b^{\bowtie}(\gamma_{\text{CONST}}(a))) = \begin{cases} \perp & \text{if } a = \perp \\ \vec{a} & \text{if } a = \vec{a}, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \{k\}, k \bowtie 0 \\ \perp & \text{if } a = \vec{a}, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \{k\}, k \not\bowtie 0 \\ \vec{a} & \text{if } a = \vec{a}, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(\vec{a})) = \mathbb{Z} \end{cases}$$

$$\alpha_{\text{CONST}}(t_b^-(\gamma_{\text{CONST}}(a))) = \begin{cases} \perp & \text{if } a = \perp \\ a & \text{if } a \neq \perp, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(a)) = \{k\}, k = 0 \\ \perp & \text{if } a \neq \perp, \llbracket \sum_{i=1}^n m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(a)) = \{k\}, k \neq 0 \\ \vec{a}[a_j/k] & \text{if } a \neq \perp, \exists j \in [1, n]. m_j \neq 0, a_j = \top, \llbracket \sum_{i \neq j} m_i x_i + b \rrbracket(\gamma_{\text{CONST}}(a)) = \{k\} \\ a & \text{otherwise} \end{cases}$$

Summing up, by Corollaries 3.4 and 3.5 we have shown the following result for the class $\mathcal{C}_{\text{CONST}}$ of non-deterministic programs with (possibly parallel) affine assignments and (conjunctive or disjunctive) linear Boolean guards.

Theorem 4.1 (Decidability and Synthesis of Inductive Invariants in CONST). *The Monniaux problem (5) on CONST for programs in $\mathcal{C}_{\text{CONST}}$, r.e. sets of initial states and recursive sets of state properties is decidable. Moreover, the algorithm AINV of Corollary 3.5 instantiated to $\text{post}_{\mathcal{P}}$ for programs $\mathcal{P} \in \mathcal{C}_{\text{CONST}}$, r.e. sets of initial states and safety properties $P \in \dot{\gamma}_{\text{CONST}}(\text{CONST}^{|\mathcal{Q}_{\mathcal{P}}|})$ synthesizes the least inductive invariant of \mathcal{P} in CONST, when this exists.*

This result could be easily extended to polynomial assignments and Boolean guards. We do not discuss the details of these extensions since our main goal here was to illustrate on the simple example of CONST how the abstract inductive invariant principle can be applied to derive decidability results and synthesis algorithms for abstract inductive invariants.

Example 4.2. Let us consider the program \mathcal{P} represented in Fig. 1 with $\Sigma_0 = \{q_1\} \times \mathbb{Z}^2$ and the valid property $P^\sharp = \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, \top) \rangle, \langle q_4, (\top, \top) \rangle \rangle \in \text{CONST}_2^{|\mathcal{Q}|}$ representing that the variable x_2 is constantly equal to 2 at the program point q_2 . It is easy to check that the algorithm AINV of Corollary 3.5 yields

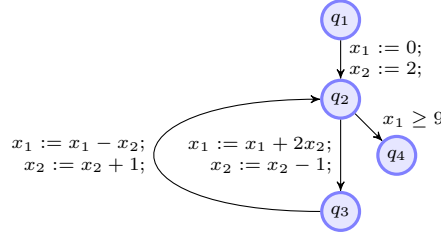


Fig. 1. A CFG representing a nondeterministic program \mathcal{P} .

the following sequence of abstract values $J^{k+1} = \dot{\alpha}_{\text{CONST}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(J^k))) \in \text{CONST}_2^{|Q|}$:

$$\begin{aligned}
 J^0 &= \dot{\alpha}_{\text{CONST}}(\Sigma_0) = \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\perp, \perp) \rangle, \langle q_3, (\perp, \perp) \rangle, \langle q_4, (\perp, \perp) \rangle \rangle \\
 J^1 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (0, 2) \rangle, \langle q_3, (\perp, \perp) \rangle, \langle q_4, (\perp, \perp) \rangle \rangle \\
 J^2 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (0, 2) \rangle, \langle q_3, (4, 1) \rangle, \langle q_4, (\perp, \perp) \rangle \rangle \\
 J^3 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (4, 1) \rangle, \langle q_4, (\perp, \perp) \rangle \rangle \\
 J^4 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, 1) \rangle, \langle q_4, (\top, 2) \rangle \rangle = \\
 &= \dot{\alpha}_{\text{CONST}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(J^4))) \dot{\leq}_{\text{CONST}} P^\#
 \end{aligned}$$

Thus, the output J^4 is the least (*i.e.* strongest) inductive invariant in CONST which allows us to prove $P^\#$. \square

Relationship with Completeness. It is easy to check that all the transfer functions t_a of affine assignments are pointwise complete, namely, $\alpha_{\text{CONST}} \circ t_a = \alpha_{\text{CONST}} \circ t_a \circ \gamma_{\text{CONST}} \circ \alpha_{\text{CONST}}$ holds. Consequently, as recalled in Section 3.2, from pointwise completeness one obtains fixpoint completeness, that is, for all unguarded programs \mathcal{P} with (parallel) affine assignments,

$$\dot{\alpha}_{\text{CONST}}(\text{lfp}(\lambda \vec{X}. \Sigma_0 \cup \text{post}_{\mathcal{P}}(\vec{X}))) = \text{lfp}(\lambda \vec{a}. \dot{\alpha}_{\text{CONST}}(\Sigma_0) \dot{\sqcup}_{\text{CONST}} \dot{\alpha}_{\text{CONST}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(\vec{a}))))).$$

However, fixpoint completeness is lost as soon as affine linear Boolean guards are taken into account, because, in general, linear guards are not pointwise complete: for example, with $n = 2$, we have that:

$$\begin{aligned}
 \alpha_{\text{CONST}}(t_{x_1=0}(\{(1, 0), (-1, 0)\})) &= \alpha_{\text{CONST}}(\emptyset) = \perp \quad \text{while} \\
 \alpha_{\text{CONST}}(t_{x_1=0}(\gamma_{\text{CONST}}(\alpha_{\text{CONST}}(\{(1, 0), (-1, 0)\})))) &= \alpha_{\text{CONST}}(t_{x_1=0}(\mathbb{Z} \times \{0\})) \\
 &= \alpha_{\text{CONST}}(\{(0, 0)\}) = \langle 0, 0 \rangle
 \end{aligned}$$

It is therefore worth remarking that the decidability result in CONST for the programs in $\mathcal{C}_{\text{CONST}}$ of Theorem 4.1 holds even if fixpoint completeness on CONST for all the programs in $\mathcal{C}_{\text{CONST}}$ does not hold.

4.2 Karr's Affine Equalities Domain

Program analysis on the domain of affine equalities has been introduced in 1976 by Karr [19] who designed some algorithms which compute for each program point the valid affine equalities between program variables. This abstract domain, here denoted by AFF , is relatively simple, precise and is widely used in numerical program analysis (see, *e.g.*, [25, 36]). Müller-Olm and Seidl [30] put forward simpler and more efficient algorithms for the transfer functions of AFF and proved that Karr's analysis algorithm is fixpoint complete for unguarded

nondeterministic affine programs, while for linearly guarded nondeterministic affine programs it is undecidable whether a given affine equality holds at a program point or not.

Let us briefly recall the definition of the abstract domain AFF , where here the n program variables assume rational values, that is, $\mathbb{V} = \mathbb{Q}$. The abstract invariants in AFF are finite (possibly empty) conjunctions of affine equalities between variables in Var_n , namely, $\bigwedge_{j=1}^k (\sum_{i=1}^n m_{i,j} x_i + b_j = 0)$. The geometric view of AFF is based on the affine hull of a subset $X \in \wp(\mathbb{Q}^n)$ which is defined by:

$$\text{aff}(X) \triangleq \{ \sum_{j=0}^m \lambda_j \vec{x}_j \in \mathbb{Q}^n \mid m \in \mathbb{N}, \lambda_j \in \mathbb{Q}, \vec{x}_j \in X, \sum_{j=0}^m \lambda_j = 1 \}.$$

This map $\text{aff} : \wp(\mathbb{Q}^n) \rightarrow \wp(\mathbb{Q}^n)$ is an upper closure on $\langle \wp(\mathbb{Q}^n), \subseteq \rangle$ whose fixpoints are the affine subspaces in \mathbb{Q}^n and therefore defines the affine equalities domain $\text{AFF} \triangleq \langle \text{aff}(\wp(\mathbb{Q}^n)), \subseteq \rangle$. Thus, any conjunction of affine equalities can be represented by an affine subspace, and vice versa. $\langle \text{aff}(\wp(\mathbb{Q}^n)), \subseteq \rangle$ is closed under arbitrary intersections, because aff is a uco, but not under unions, i.e., aff is not an additive uco. The lub in AFF of a set of affine subspaces $\mathcal{X} \subseteq \text{AFF}$ is given by $\sqcup_{\text{AFF}} \mathcal{X} \triangleq \text{aff}(\cup_{X \in \mathcal{X}} X)$. A complex algorithm for computing a binary lub $A \sqcup_{\text{AFF}} A'$ of two affine subspaces $A, A' \in \text{AFF}$ was given by Karr [19, Section 5.2], a simpler and more efficient version is described by Miné [25, Section 5.2.2]. $\langle \text{AFF}, \subseteq \rangle$ is a complete lattice of finite height $n + 1$, because if $A, A' \in \text{AFF}$ and $A \subsetneq A'$ then $\dim(A) < \dim(A')$, where $\dim(\emptyset) = -1$ and $\dim(\mathbb{Q}^n) = n$.

Karr gave already in [19, Section 4.2] an algorithm for computing the bca of an affine assignment $t_a^j \equiv x_j := \sum_{i=1}^n m_i x_i + b$, with $j \in [1, n]$. Karr's algorithm is based on representing affine subspaces by kernels of affine transformations (i.e., matrices) and distinguishes between invertible (such as $x_i := x_i + k$) and noninvertible (such as $x_i := x_j + k$) affine assignments. Müller-Olm and Seidl [30] put forward a more efficient algorithm which is based on representing affine subspaces by affine bases. It is worth remarking that [30, Lemma 2] also observes that the bca of t_a^j is pointwise complete, namely $\text{aff} \circ t_a^j \circ \text{aff} = \text{aff} \circ t_a^j$ holds. We are not interested here in the details and complexities of these algorithms which implement the best abstraction in AFF of the affine assignments t_a^j , since here we just exploit the fact that the best correct approximation $\text{aff} \circ t_a^j : \text{AFF} \rightarrow \text{AFF}$ is computable. In turn, computability of parallel affine assignments $\vec{x} := M\vec{x} + \vec{b}$ easily follows.

Müller-Olm and Seidl [30] also show that the bca of a nondeterministic assignments $t_{a?}^j \triangleq x_j := ?$ is computable, where for all $Y \in \wp(\mathbb{Q}^n)$, the corresponding transfer function is defined by: $t_{x_j:=?}(Y) \triangleq \{ \vec{y} \mid y_j/v \in \mathbb{Q}^n \mid \vec{y} \in Y, v \in \mathbb{Q} \}$. An observation in [30, Lemma 4] states that $\text{aff}(t_{x_j:=?}(\text{aff}(Y))) = \text{aff}(t_{x_j:=0}(\text{aff}(Y))) \sqcup_{\text{AFF}} \text{aff}(t_{x_j:=1}(\text{aff}(Y)))$, thus reducing the computation of $\text{aff}(t_{x_j:=?}(\text{aff}(Y)))$ to computing the lub of the bca's of the transfer functions of two single affine assignments $x_j := 0$ and $x_j := 1$.

As already shown by Karr [19, Section 4.1] (see also [25, Section 5.2.3] for a modern approach), it turns out that best correct approximations of affine equalities Boolean guards of the shape $t_b^- \triangleq \sum_{i=1}^n m_i x_i + b = 0$ are computable. However, no algorithm for computing the bca of a generic negation $t_b^{\neq} \triangleq \sum_{i=1}^n m_i x_i + b \neq 0$ is known in literature. Both Karr [19, Section 4.1] and Miné [25, Section 5.2.3] propose to soundly approximate a negated affine equality guard t_b^{\neq} simply by the identity transfer function, meaning that t_b^{\neq} is replaced by the *true* predicate. Karr [19, Section 4.1] mentions that “a general study of how best to handle decision nodes which are not of the simple form t_b^- is in preparation”, but this document never appeared. To the best of our knowledge, the literature provides no algorithm for computing the bca of t_b^{\neq} in AFF , and we conjecture that, in general, it could not be computable.

Summing up, as a consequence of Corollaries 3.4 and 3.5 we therefore obtain the following result for the class \mathcal{C}_{AFF} of nondeterministic programs with (possibly parallel) affine assignments, (possibly parallel) nondeterministic assignments and (conjunctive or disjunctive) positive linear equalities guards.

Theorem 4.3 (Decidability and Synthesis of Inductive Invariants in AFF). *The Monniaux problem (5) on AFF for programs in \mathcal{C}_{AFF} , i.e. sets of initial states and recursive sets of state properties is decidable. Moreover, the algorithm AINV of Corollary 3.5 instantiated to $\text{post}_{\mathcal{P}}$ for programs $\mathcal{P} \in \mathcal{C}_{\text{AFF}}$, i.e. sets of initial states and safety properties $P \in \text{AFF}$ synthesizes the least inductive invariant of \mathcal{P} in AFF , when this exists.*

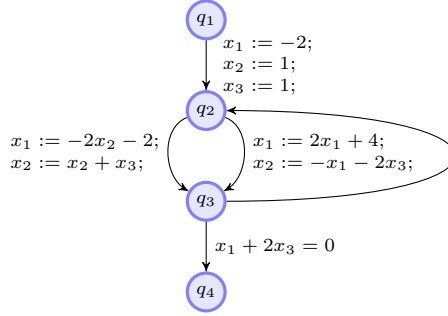


Fig. 2. A nondeterministic program \mathcal{R} .

Example 4.4. Let us consider the nondeterministic program \mathcal{R} in Fig. 2 with $\Sigma_0 = \{q_1\} \times \mathbb{Q}^3$ and the property $P^\# = \langle \langle q_1, \top \rangle, \langle q_2, \top \rangle, \langle q_3, \top \rangle, \langle q_4, x_1 + x_2 + 1 = 0 \rangle \rangle \in \text{AFF}^{|\mathcal{Q}|}$. The algorithm AINV of Corollary 3.5 yields the following sequence of abstract values $I^j \in \text{AFF}^{|\mathcal{Q}|}$:

$$\begin{aligned}
 I^0 &= \dot{\alpha}_{\text{AFF}}(\Sigma_0) = \langle \langle q_1, \top \rangle, \langle q_2, \perp \rangle, \langle q_3, \perp \rangle, \langle q_4, \perp \rangle \rangle \\
 I^1 &= \langle \langle q_1, \top \rangle, \langle q_2, x_1 + 2 = 0 \wedge x_2 - 1 = 0 \wedge x_3 - 1 = 0 \rangle, \langle q_3, \perp \rangle, \langle q_4, \perp \rangle \rangle \\
 I^2 &= \langle \langle q_1, \top \rangle, \langle q_2, x_1 + 2 = 0 \wedge x_2 - 1 = 0 \wedge x_3 - 1 = 0 \rangle, \\
 &\quad \langle q_3, x_1 + 2x_2 = 0 \wedge x_3 = 1 \rangle, \langle q_4, \perp \rangle \rangle \\
 I^3 &= \langle \langle q_1, \top \rangle, \langle q_2, x_1 + 2x_2 = 0 \wedge x_3 = 1 \rangle, \langle q_3, x_1 + 2x_2 = 0 \wedge x_3 = 1 \rangle, \\
 &\quad \langle q_4, x_1 + x_2 + 1 = 0 \wedge x_3 = 1, \perp \rangle \rangle = \dot{\alpha}_{\text{AFF}}(\text{post}_{\mathcal{R}}(\dot{\gamma}_{\text{AFF}}(I^3))) \dot{\leq}_{\text{AFF}} P^\#
 \end{aligned}$$

The output I^3 is the analysis of \mathcal{R} with the bca's of its transfer functions in AFF , *i.e.*, it is the least inductive invariant in AFF which allows us to prove that $P^\#$ holds. \square

Relationship with Completeness. Müller-Olm and Seidl [30] implicitly show that the transfer functions of affine assignments t_a and of nondeterministic assignments $t_{x_j:=?}$ are pointwise complete. In fact, [30, Lemma 2] shows that for all $X \in \wp(\mathbb{Q}^n)$, $t_a(\text{aff}(X)) = \text{aff}(t_a(X))$, from which we easily obtain:

$$\begin{aligned}
 \text{aff}(t_a(X)) &= \text{aff}(\text{aff}(t_a(X))) = \text{aff}(t_a(\text{aff}(X))) \\
 \text{aff}(t_{x_j:=?}(X)) &= \text{aff}(\cup_{z \in \mathbb{Q}} t_{x_j:=z}(X)) = \text{aff}(\cup_{z \in \mathbb{Q}} \text{aff}(t_{x_j:=z}(X))) = \\
 &= \text{aff}(\cup_{z \in \mathbb{Q}} \text{aff}(t_{x_j:=z}(\text{aff}(X)))) = \text{aff}(\cup_{z \in \mathbb{Q}} t_{x_j:=z}(\text{aff}(X))) = \text{aff}(t_{x_j:=?}(\text{aff}(X)))
 \end{aligned}$$

Thus, since pointwise completeness entails fixpoint completeness, similarly to Section 4.1, for all unguarded programs \mathcal{P} with affine and nondeterministic assignments, $\dot{\alpha}_{\text{AFF}}(\text{lfp}(\lambda \vec{X}. \Sigma_0 \cup \text{post}_{\mathcal{P}}(\vec{X}))) = \text{lfp}(\lambda \vec{a}. \dot{\alpha}_{\text{AFF}}(\Sigma_0) \dot{\sqcup}_{\text{AFF}} \dot{\alpha}_{\text{AFF}}(\text{post}_{\mathcal{P}}(\dot{\gamma}_{\text{AFF}}(\vec{a}))))$ holds. Here, fixpoint completeness is lost as soon as affine equality guards are included (these programs still belong to \mathcal{C}_{AFF}), because they are not pointwise complete, *e.g.*:

$$\begin{aligned}
 \text{aff}(t_{x_1=0}(\{(1,0), (-1,0)\})) &= \text{aff}(\emptyset) = \emptyset \quad \text{while} \\
 \text{aff}(t_{x_1=0}(\text{aff}(\{(1,0), (-1,0)\}))) &= \text{aff}(t_{x_1=0}(x_2 = 0)) = \text{aff}(\{(0,0)\}) = \{(0,0)\}
 \end{aligned}$$

Moreover, a result in [30, Section 7] proves that once affine equality guards are added to nondeterministic affine programs it becomes undecidable whether a given affine equality holds in some program point or not. This undecidability does not prevent the decidability result of Theorem 4.3, since these two decision problems are orthogonal.

5 Co-Inductive Synthesis of Abstract Inductive Invariants

In the following we design a synthesis algorithm which, by generalizing a recent algorithm by Padon et al. [32], outputs the *most abstract* inductive invariant in an abstract domain, when this exists. This algorithm is obtained by dualizing the procedure AINV in Corollary 3.5 to a co-inductive greatest fixpoint computation and will require that the abstract domain is equipped with a suitable well-quasiorder relation. Let us recall that a qoset D_{\leq} is a well-quasiordered set (wqoset), and \leq is called well-quasiorder (wqo), when for every countably infinite sequence of elements $\{x_i\}_{i \in \mathbb{N}}$ in D there exist $i, j \in \mathbb{N}$ such that $i < j$ and $x_i \leq x_j$. Equivalently, D is a wqoset iff D is DCC (also called well-founded) and D has no infinite antichain.

Let $\mathcal{T} = \langle \Sigma, \tau \rangle$ be a transition system whose successor transformer is post , so that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X)) \in \wp(\Sigma)$ are the reachable states of \mathcal{T} from some set of initial states $\Sigma_0 \in \wp(\Sigma)$. [32] considers abstract invariants ranging in a set (of semantics of logical formulae) $L \subseteq \wp(\Sigma)$ and assumes (in [32, Theorem 4.2]) that $\langle L, \subseteq \rangle$ is closed under finite intersections (*i.e.*, logical conjunctions). Accordingly to Assumption 3.1, we ask that $\langle L, \subseteq \rangle$ satisfies the requirement of being an abstract domain of the concrete domain $\langle \wp(\Sigma), \subseteq \rangle$, which corresponds to ask that $\langle L, \subseteq \rangle$ is closed under arbitrary, rather than finite, intersections. Thus, L is the image of an upper closure $\check{\mu}_L \in \text{uco}(\wp(\Sigma), \subseteq)$ defined by: $\check{\mu}_L(X) \triangleq \cap \{\phi \in L \mid X \subseteq \phi\}$.

The three key definitions and related assumptions of the synthesis algorithm defined in [32, Theorem 4.2] concern a quasi-order $\sqsubseteq_L \subseteq \Sigma \times \Sigma$ between states, a function $Av_L : \Sigma \rightarrow \wp(\Sigma)$ called Avoid, and an abstract transition relation $\tau^L \subseteq \Sigma \times \Sigma$:

- | | |
|--|--|
| (1) $s \sqsubseteq_L s' \triangleq \forall \phi \in L. s' \in \phi \Rightarrow s \in \phi$ | Assumption (A ₁): $\langle \Sigma, \sqsubseteq_L \rangle$ is a wqoset |
| (2) $Av_L(s) \triangleq \cup \{\phi \in L \mid \phi \subseteq \neg\{s\}\}$ | Assumption (A ₂): $\forall s \in \Sigma. Av_L(s) \in L$ |
| (3) $(s, s') \in \tau^L \triangleq (s, s') \in \tau \vee s' \sqsubseteq_L s$ | Abstract Transition System $\mathcal{T}^L \triangleq \langle \Sigma, \tau^L \rangle$ |

Correspondingly, we define the down-closure $\delta_L : \wp(\Sigma) \rightarrow \wp(\Sigma)$ of \sqsubseteq_L , we generalize $Av_L : \wp(\Sigma) \rightarrow \wp(\Sigma)$ to sets of states and we define the successor transformer $\text{post}^L : \wp(\Sigma) \rightarrow \wp(\Sigma)$ of \mathcal{T}^L as follows:

- | | |
|---|--|
| (1) $\delta_L(X) \triangleq \{s \in \Sigma \mid \exists s' \in X. s \sqsubseteq_L s'\}$ | Down-closure of the qo \sqsubseteq_L |
| (2) $Av_L(X) \triangleq \cup \{\phi \in L \mid \phi \subseteq \neg X\}$ | Av_L for any set X of states |
| (3) $\text{post}^L(X) \triangleq \text{post}(X) \cup \delta_L(X)$ | Successor transformer of \mathcal{T}^L |

Lemma 5.1. *The following properties hold:*

- (a) $s \sqsubseteq_L s'$ iff $\check{\mu}_L(\{s\}) \subseteq \check{\mu}_L(\{s'\})$.
- (b) (A₁) holds iff $\langle L, \subseteq \rangle$ is a well-quasi order.
- (c) (A₂) holds iff L is closed under arbitrary unions.
- (d) If (A₂) holds then $\delta_L = \check{\mu}_L$ and both are additive ucos (and $\delta_L(\phi) = \phi \Leftrightarrow \phi \in L$).
- (e) For all $\Sigma_0 \in \wp(\Sigma)$, $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}^L(X)) = \text{lfp}(\lambda X. \check{\mu}_L(\Sigma_0 \cup \text{post}(X)))$.

Proof. (a) If $s \sqsubseteq_L s'$ and $t \in \check{\mu}_L(\{s\})$ then $t \in \check{\mu}_L(\{s'\}) = \cap \{\phi \in L \mid s' \in \phi\}$: if $\phi \in L$ and $s' \in \phi$ then $s \in \phi$, so that, since $t \in \check{\mu}_L(\{s\})$, $t \in \phi$. Conversely, if $\check{\mu}_L(\{s\}) \subseteq \check{\mu}_L(\{s'\})$, $\phi \in L$ and $s' \in \phi$, then, since $s \in \check{\mu}_L(\{s'\})$, $s \in \phi$.

(b) By (a), we equivalently prove that $\langle \{\check{\mu}_L(\{s\}) \mid s \in \Sigma\}, \subseteq \rangle$ is a wqo iff $\langle L, \subseteq \rangle$ is a wqo.

(\Rightarrow): [32, Lemma 4.6] proves $\langle L, \subseteq \rangle$ is well-founded, we additionally show that it does not contain infinite antichains. By contradiction, assume that $\{\phi_i\}_{i \in \mathbb{N}}$ is an infinite antichain in $\langle L, \subseteq \rangle$. Thus, for all $i \neq j$, $\phi_i \not\subseteq \phi_j$ and $\phi_j \not\subseteq \phi_i$, so that there exist $s_{i,j} \in \phi_i \setminus \phi_j$ and $s_{j,i} \in \phi_j \setminus \phi_i$. From $s_{j,i} \in \phi_j$ we obtain that $\check{\mu}_L(\{s_{j,i}\}) \subseteq \check{\mu}_L(\phi_j) = \phi_j$. From $s_{i,j} \notin \phi_j$, we obtain that $s_{i,j} \in \check{\mu}_L(\{s_{i,j}\}) \not\subseteq \phi_{j,i}$. It turns out that $\check{\mu}_L(\{s_{i,j}\}) \not\subseteq \check{\mu}_L(\{s_{j,i}\})$, otherwise from $s_{i,j} \in \check{\mu}_L(\{s_{i,j}\}) \subseteq \check{\mu}_L(\{s_{j,i}\}) \subseteq \phi_j$ we would obtain the contradiction $s_{i,j} \in \phi_j$.

Dually, $\check{\mu}_L(\{s_{j,i}\}) \not\subseteq \check{\mu}_L(\{s_{i,j}\})$ holds. Thus, for any $i \in \mathbb{N}$, $\{\check{\mu}_L(\{s_{i,j}\}) \mid j \in \mathbb{N}, j \neq i\}$ is an infinite antichain in $\langle \check{\mu}_L(\{s\}) \mid s \in \Sigma \rangle, \subseteq$, which is a contradiction.

(\Leftarrow): $\langle \check{\mu}_L(\{s\}) \mid s \in \Sigma \rangle, \subseteq$ is trivially a wqo because $\{\check{\mu}_L(\{s\}) \mid s \in \Sigma\} \subseteq L$ and L is a wqo.

(c) Assume that for all $s \in \Sigma$, $Av_L(s) \in L$. Let us show that for all $S \in \wp(\Sigma)$, $\cap_{s \in S} Av_L(s) = Av_L(S)$.

(\supseteq): Let $t \in \phi$ for some $\phi \in L$ such that $\phi \subseteq \neg S$. Then, for all $s \in S$, $\phi \subseteq Av_L(s)$, so that $t \in \cap_{s \in S} Av_L(s)$.

(\subseteq): Let $t \in \cap_{s \in S} Av_L(s)$. For all $s \in S$, there exists $\phi_s \in L$ such that $\phi_s \subseteq \neg\{s\}$ and $t \in \phi_s$. Thus, $\cap_{s \in S} \phi_s \in L$ and $t \in \cap_{s \in S} \phi_s \subseteq \neg S$, meaning that $t \in Av_L(S)$.

Thus, since L is assumed to be closed under arbitrary intersections we obtain that $Av_L(S) = \cap_{s \in S} Av_L(s) \in L$. Consider now $\Phi \subseteq L$. Then, $Av_L(\neg(\cup \Phi)) = \cup\{\phi \in L \mid \phi \subseteq \neg\neg(\cup \Phi)\} = \cup\{\phi \in L \mid \phi \subseteq \cup \Phi\} = \cup \Phi$, so that $\cup \Phi \in L$.

Conversely, if L is closed under arbitrary unions then $Av_L(s) = \cup\{\phi \in L \mid \phi \subseteq \neg\{s\}\} \in L$.

(d) Since \sqsubseteq_L is a quasi-order relation, its down-closure δ_L is an upper closure on $\langle \wp(\Sigma), \subseteq \rangle$. By (a), $\delta_L(X) = \{s \in \Sigma \mid \exists s' \in X. s \sqsubseteq_L s'\} = \{s \in \Sigma \mid \exists s' \in X. \check{\mu}_L(\{s\}) \subseteq \check{\mu}_L(\{s'\})\} = \{s \in \Sigma \mid \exists s' \in X. s \in \check{\mu}_L(\{s'\})\} = \cup_{s' \in X} \check{\mu}_L(\{s'\})$. By (c), since L is closed under arbitrary unions, the upper closure $\check{\mu}_L$ is additive, so that $\cup_{s' \in X} \check{\mu}_L(\{s'\}) = \check{\mu}_L(\cup_{s' \in X} \{s'\}) = \check{\mu}_L(X)$, consequently $\delta_L(X) = \check{\mu}_L(X)$. In particular, $\delta_L(\phi) = \phi \Leftrightarrow \check{\mu}_L(\phi) = \phi \Leftrightarrow \phi \in L$.

(e) By Lemma 5.1 (d), $\text{lfp}(\lambda X. \text{post}(X) \cup \delta_L(X) \cup \Sigma_0) = \text{lfp}(\lambda X. \text{post}(X) \cup \check{\mu}_L(X) \cup \Sigma_0)$. It turns out that

$$\begin{aligned} \Sigma_0 \cup \text{post}(X) \cup \delta_L(X) &\subseteq X \Leftrightarrow \text{[by Lemma 5.1 (d)]} \\ \Sigma_0 \cup \text{post}(X) \cup \check{\mu}_L(X) &\subseteq X \Leftrightarrow \text{[by set theory]} \\ \Sigma_0 \subseteq X \wedge \text{post}(X) &\subseteq X \wedge \check{\mu}_L(X) \subseteq X \Leftrightarrow \text{[as } \check{\mu}_L \text{ is a uco]} \\ \Sigma_0 \subseteq X \wedge \text{post}(X) &\subseteq X \wedge \check{\mu}_L(X) = X \Leftrightarrow \text{[as } \check{\mu}_L(X) = X\text{]} \\ \Sigma_0 \subseteq X \wedge \text{post}(\check{\mu}_L(X)) &\subseteq X \wedge \check{\mu}_L(X) = X \Leftrightarrow \text{[by set theory]} \\ \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X) &\subseteq X \end{aligned}$$

Thus, by Knaster-Tarski theorem, $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) = \text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X) \cup \delta_L(X))$. We also have that:

$$\begin{aligned} \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X) &\subseteq X \Leftrightarrow \text{[by the equivalences above]} \\ \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X) &\subseteq X = \check{\mu}_L(X) \Leftrightarrow \text{[by set theory]} \\ \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) &\subseteq X = \check{\mu}_L(X) \Leftrightarrow \text{[as } \check{\mu}_L(X) = X\text{]} \\ \Sigma_0 \cup \text{post}(X) &\subseteq X = \check{\mu}_L(X) \Leftrightarrow \text{[as } \check{\mu}_L \text{ is a uco]} \\ \check{\mu}_L(\Sigma_0 \cup \text{post}(X)) &\subseteq X = \check{\mu}_L(X) \Leftrightarrow \text{[by set theory]} \\ \check{\mu}_L(\Sigma_0 \cup \text{post}(X)) &\subseteq X \end{aligned}$$

Thus, by Knaster-Tarski theorem, $\text{lfp}(\check{\mu}_L(\Sigma_0 \cup \text{post}(X))) \subseteq \text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X))$ follows. Moreover, if $F \triangleq \text{lfp}(\check{\mu}_L(\Sigma_0 \cup \text{post}(X)))$, so that $F = \check{\mu}_L(F) = \Sigma_0 \cup \text{post}(F)$, then $\Sigma_0 \cup \text{post}(\check{\mu}_L(F)) \cup \check{\mu}_L(F) = \Sigma_0 \cup \text{post}(F) \cup \check{\mu}_L(F) = F$, and this implies that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) \subseteq \text{lfp}(\check{\mu}_L(\Sigma_0 \cup \text{post}(X)))$. Therefore, $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) = \text{lfp}(\check{\mu}_L(\Sigma_0 \cup \text{post}(X)))$. \square

In particular, Lemma 5.1 (e) states that the set of reachable states in the abstract transition system \mathcal{T}^L coincides with the set of reachable states of the abstract transition system $\mathcal{T}^{\check{\mu}_L} \triangleq \langle \Sigma, \check{\mu}_L \circ \text{post}_{\mathcal{T}} \rangle$ obtained by considering the best correct approximation of $\text{post}_{\mathcal{T}}$ in the abstract domain $\check{\mu}_L$. As a direct consequence of the abstract inductive invariant principle Lemma 3.2 (a), we obtain the following characterization of the abstract inductive invariants ranging in L of the transition system \mathcal{T} which relies on the reachable states of its best abstraction $\mathcal{T}^{\check{\mu}_L}$ in the domain $\check{\mu}_L$.

Corollary 5.2. *Let $\Sigma_0, P \in \wp(\Sigma)$. Then, $\exists \phi \in L. \Sigma_0 \subseteq \phi \wedge \text{post}(\phi) \subseteq \phi \wedge \phi \subseteq P$ iff $\text{Reach}[\mathcal{T}^{\check{\mu}_L}, \Sigma_0] \subseteq P$.*

Proof. It turns out that

$$\begin{aligned}
\text{lfp}(\lambda X. \check{\mu}_L(\Sigma_0 \cup \text{post}(X))) &\subseteq P \Leftrightarrow [\text{by Lemma 3.2 (a) for ucos}] \\
\exists \phi \in \check{\mu}_L(\wp(\Sigma)). \Sigma_0 \cup \text{post}(\phi) &\subseteq \phi \wedge \phi \subseteq P \Leftrightarrow [\text{as } \check{\mu}_L(\wp(\Sigma)) = L] \\
\exists \phi \in L. \Sigma_0 &\subseteq \phi \wedge \text{post}(\phi) \subseteq \phi \wedge \phi \subseteq P
\end{aligned}
\quad \square$$

5.1 Co-Inductive Invariants

Following [32], in the following we make assumption (A₂), that is, by Lemma 5.1 (c), we assume that $L \subseteq \wp(\Sigma)$ is closed under arbitrary unions. This means that $\check{\mu}_L$ is an additive uco on $\wp(\Sigma)_{\subseteq}$, *i.e.*, in abstract interpretation terminology, $\check{\mu}_L$ is a *disjunctive* abstract domain whose abstract lub does not lose precision (see, *e.g.*, [25, Section 6.3]). Furthermore, we also have that L is the image of a co-additive (*i.e.*, preserving arbitrary intersections) lower closure $\hat{\mu}_L : \wp(\Sigma) \rightarrow \wp(\Sigma)$ defined by $\hat{\mu}_L(X) \triangleq \bigcup \{\phi \in L \mid \phi \subseteq X\}$. It turns out that the uco $\check{\mu}_L$ is adjoint to the lco $\hat{\mu}_L$ (this is called abstract adjointness in [2, Section 3.5]), namely, $\check{\mu}_L(X) \subseteq Y \Leftrightarrow X \subseteq \hat{\mu}_L(Y)$. In fact, if $\check{\mu}_L(X) \subseteq Y$ then, by applying $\hat{\mu}_L$, $X \subseteq \check{\mu}_L(X) = \hat{\mu}_L(\check{\mu}_L(X)) \subseteq \hat{\mu}_L(Y)$; the converse is dual.

As observed in [2, Theorem 4], the inductive invariant principle (2) can be dualized when f admits right-adjoint $\tilde{f} : C \rightarrow C$ (this happens iff f is additive): in this case,

$$\text{lfp}(\lambda x. c \vee f(x)) \leq c' \Leftrightarrow c \leq \text{gfp}(\lambda x. c' \wedge \tilde{f}(x)) \quad (7)$$

holds and one obtains a *co-inductive invariant principle*:

$$c \leq \text{gfp}(\lambda x. \tilde{f}(x) \wedge c') \Leftrightarrow \exists j \in C. c \leq j \wedge j \leq \tilde{f}(j) \wedge j \leq c' \quad (8)$$

One such $j \in C$ is therefore called a *co-inductive invariant* of f for $\langle c, c' \rangle$.

The co-inductive invariant proof method (8) can be applied to safety verification of any transition system \mathcal{T} because post is additive and therefore it always admits right adjoint $\widetilde{\text{pre}}$ (cf. Section 2). Hence, we obtain that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X)) \subseteq P$ iff $\Sigma_0 \subseteq \text{gfp}(\lambda X. \widetilde{\text{pre}}(X) \cap P)$ iff there exists a co-inductive invariant for $\widetilde{\text{pre}}$ for $\langle \Sigma_0, P \rangle$. By (2) and (8), it turns out that I is an inductive invariant of post for $\langle \Sigma_0, P \rangle$ iff I is a co-inductive invariant of $\widetilde{\text{pre}}$ for $\langle \Sigma_0, P \rangle$. Also, while $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X))$ is the least, *i.e.* logically strongest, inductive invariant, we have that $\text{gfp}(\lambda X. \widetilde{\text{pre}}(X) \cap P)$ is the greatest, *i.e.* logically weakest, inductive invariant [2, Theorem 6].

We show how the co-inductive invariant principle (8) applied to the best abstract transition system $\mathcal{T}^{\check{\mu}_L} = (\Sigma, \check{\mu}_L \circ \text{post}_{\mathcal{T}})$ provides exactly the synthesis algorithm by Padon et al. [32, Algorithm 1]. In order to do this, we first observe the following alternative characterization of the reachable states of $\mathcal{T}^{\check{\mu}_L}$.

Lemma 5.3. $\text{lfp}(\lambda X. \Sigma_0 \cup \check{\mu}_L(\text{post}(X))) = \text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)).$

Proof. The proof of Lemma 5.1 (e) shows that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) = \text{lfp}(\check{\mu}_L(\Sigma_0 \cup \text{post}(X)))$. Moreover, since $\check{\mu}_L$ is additive and $\check{\mu}_L(\Sigma_0) = \Sigma_0$, we also have that $\check{\mu}_L(\Sigma_0 \cup \text{post}(X)) = \check{\mu}_L(\Sigma_0) \cup \check{\mu}_L(\text{post}(X)) = \Sigma_0 \cup \check{\mu}_L(\text{post}(X))$, and this allows us to conclude. \square

Consequently, $\text{lfp}(\lambda X. \Sigma_0 \cup \check{\mu}_L(\text{post}(X))) \subseteq P \Leftrightarrow \text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) \subseteq P$ holds. Since $\lambda X. \text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)$ is additive, we can apply the co-inductive invariant principle (8) by considering its adjoint function, which is as follows:

$$\begin{aligned}
\text{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X) &\subseteq Y \Leftrightarrow \text{post}(\check{\mu}_L(X)) \subseteq Y \wedge \check{\mu}_L(X) \subseteq Y \Leftrightarrow \\
X &\subseteq \hat{\mu}_L(\widetilde{\text{pre}}(Y)) \wedge X \subseteq \hat{\mu}_L(Y) \Leftrightarrow X \subseteq \hat{\mu}_L(\widetilde{\text{pre}}(Y)) \cap \hat{\mu}_L(Y)
\end{aligned}$$

Thus, by Lemma 5.3 and (7), we obtain:

$$\begin{aligned} \text{lfp}(\lambda X. \check{\mu}_L(\Sigma_0) \cup \check{\mu}_L(\text{post}(X))) \subseteq P &\Leftrightarrow \check{\mu}_L(\Sigma_0) \subseteq \text{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P)) \\ &\Leftrightarrow \Sigma_0 \subseteq \text{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P)) \end{aligned}$$

and, in turn, by the abstract inductive invariant principle (Lemma 3.2 (a) for ucos) applied to $\check{\mu}_L \circ (\lambda X. \Sigma_0 \cup \text{post}(X)) = \lambda X. \check{\mu}_L(\Sigma_0) \cup \check{\mu}_L(\text{post}(X))$ we get:

$$\exists \phi \in L. \Sigma_0 \subseteq \phi \wedge \text{post}(\phi) \subseteq \phi \wedge \phi \subseteq P \Leftrightarrow \Sigma_0 \subseteq \text{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P)).$$

This leads us to use the algorithm introduced by Cousot [2, Algorithm 2] which synthesizes an inductive invariant by using a co-inductive method that applies Knaster-Tarski theorem to compute the iterates of the greatest fixpoint of $\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P)$ as long as the current iterate I_1 contains the initial states in Σ_0 :

Algorithm 1: Co-inductive backward abstract inductive invariant synthesis.

```

 $I_1 := \Sigma;$ 
while  $\Sigma_0 \subseteq I_1$  do // Loop invariant:  $I_1 \in L$ 
  if  $(I_1 = \hat{\mu}_L(\widetilde{\text{pre}}(I_1) \cap I_1 \cap P))$  then return  $I_1$  is an inductive invariant in  $L$ ;
   $I_1 := I_1 \cap \hat{\mu}_L(\widetilde{\text{pre}}(I_1) \cap I_1 \cap P);$ 
return no inductive invariant in  $L$ ;

```

Since, $\widetilde{\text{pre}}$ is computable and, by Lemma 5.1, $\langle \hat{\mu}_L, \subseteq \rangle = \langle L, \subseteq \rangle$ is a wqo, we immediately obtain that Algorithm 1 is correct and terminating. Furthermore, if Algorithm 1 outputs an inductive invariant I_1 proving the property P then I_1 is the *greatest* inductive invariant proving P . It turns out that Algorithm 1 exactly coincides with the synthesis algorithm by Padon et al. [32], which is replicated below as Algorithm 2.

Algorithm 2: Inductive invariant algorithm by [32].

```

 $I_2 := \Sigma;$ 
while  $I_2$  is not an inductive invariant do // Loop invariant:  $I_2 \in L$ 
  if  $\Sigma_0 \not\subseteq I_2$  then return no inductive invariant in  $L$ ;
  choose  $s \in \Sigma$  as a counterexample to inductiveness of  $I_2$ ;
   $I_2 := I_2 \cap \text{Av}_L(s);$ 
return  $I_2$  is an inductive invariant in  $L$ ;

```

Lemma 5.4. Let $I \in L$ and $\Sigma_0 \subseteq I$.

- (a) there exists a counterexample to inductiveness of I iff $I \not\subseteq \widetilde{\text{pre}}(I) \cap P$ iff $I \neq \hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P)$.
- (b) If $s \in \Sigma$ is a counterexample to inductiveness of I then $\hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P) \subseteq \text{Av}_L(s)$.

Proof. (a) Under the assumption that $\Sigma_0 \subseteq I$, s is a counterexample to inductiveness of I iff $(s \in I \wedge \text{post}(s) \not\subseteq I) \vee s \in I \cap \neg P$. Observe that $\text{post}(s) \not\subseteq I$ iff $s \notin \widetilde{\text{pre}}(I)$, so that $\exists s \in \Sigma. s \in I \wedge \text{post}(s) \not\subseteq I$ iff $I \not\subseteq \widetilde{\text{pre}}(I)$. Hence, $\exists s \in \Sigma. (s \in I \wedge \text{post}(s) \not\subseteq I) \vee s \in I \cap \neg P$ iff $I \not\subseteq \widetilde{\text{pre}}(I) \cap P$. Also:

$$\begin{aligned} I &= \hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P) \Leftrightarrow \quad [\text{as } I \in L] \\ I &= \hat{\mu}_L(I) = \hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P) \Leftrightarrow \quad [\text{as } \widetilde{\text{pre}}(I) \cap I \cap P \subseteq I] \\ I &= \hat{\mu}_L(I) \subseteq \hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P) \Leftrightarrow \quad [\text{as } \hat{\mu}_L \text{ is a lco}] \\ I &= \hat{\mu}_L(I) \subseteq \widetilde{\text{pre}}(I) \cap I \cap P \Leftrightarrow \\ &\quad I \subseteq \widetilde{\text{pre}}(I) \cap P \end{aligned}$$

- (b) The proof of point (a) shows that if $s \in \Sigma$ is a counterexample to inductiveness of I then $s \in I$ and $s \notin \widetilde{\text{pre}}(I) \cap P$. Then, $\widetilde{\text{pre}}(I) \cap P \subseteq \neg\{s\}$, so that, by monotonicity of $\hat{\mu}_L$, $\hat{\mu}_L(\widetilde{\text{pre}}(I) \cap P) \subseteq \hat{\mu}_L(\neg\{s\})$ and, in turn, $\hat{\mu}_L(\widetilde{\text{pre}}(I) \cap I \cap P) \subseteq \hat{\mu}_L(\neg\{s\}) = \text{Av}_L(s)$. \square

Theorem 5.5. *Algorithm 1 = Algorithm 2.*

Proof. Consider the following variation of Algorithm 1:

Algorithm 3: A modification of Algorithm 1.

```

 $I_4 := \Sigma;$ 
while  $\Sigma_0 \subseteq I_4$  do // Invariant:  $I_4 \in L$ 
  if  $(I_4 \setminus (\widetilde{\text{pre}}(I_4) \cap P) = \emptyset)$  then return  $I_4$  is an inductive invariant in  $L$ ;
  choose  $s \in I_4 \setminus (\widetilde{\text{pre}}(I_4) \cap P)$ ;
   $I_4 := I_4 \cap \hat{\mu}_L(\{s\});$ 
return no inductive invariant in  $L$ ;

```

Algorithm 1 returns $I_1 \in L$ iff $I_1 = \text{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P))$. In this case, by Lemma 5.4 (a), since $\Sigma_0 \subseteq I_1$ holds, I_1 is an (actually, the greatest) inductive invariant in L . Otherwise, Algorithm 1 returns “no inductive invariant in L ”. By Lemma 5.4 (a), Algorithm 3 returns $I_4 \in L$ iff $I_4 \subseteq \widetilde{\text{pre}}(I_4) \cap P$ iff $I_4 = \hat{\mu}_L(\widetilde{\text{pre}}(I_4) \cap I_4 \cap P)$, otherwise it returns “no inductive invariant in L ”. Algorithm 2 returns $I_2 \in L$ iff I_2 is an inductive invariant, otherwise it returns “no inductive invariant in L ”. Let I_k^n be the current candidate invariant of Algorithm $k \in \{1, 2, 4\}$ at its n -th iteration and I_k be the output invariant of Algorithm k . By Lemma 5.4 (b), $I_1^n \subseteq I_4^n = I_2^n$, so that $I_1 \subseteq I_4 = I_2$. Since I_k are fixpoints of $\lambda X. \hat{\mu}_L(\widetilde{\text{pre}}(X) \cap X \cap P)$ and I_1 is the greatest fixpoint, it turns out that $I_1 = I_4 = I_2$. \square

This shows that Algorithm 2 in [32] for a disjunctive GC-based abstract domain A amounts to a backward static analysis (*i.e.*, propagating $\widetilde{\text{pre}}$) using the best correct approximations in A of the transfer functions, as long as the ordering of A guarantees its termination, *e.g.*, because A is well-founded.

Example 5.6. It turns out that CONST, as defined in Section 4.1, is a disjunctive abstract domain with finite height and its bca’s of affine assignments and linear Boolean guards are computable. Thus, CONST satisfies the hypotheses guaranteeing the correctness and termination of Algorithm 1 (and 2). Let us apply Algorithm 1 to the program \mathcal{P} of Example 4.2 in Fig. 1, still with $\Sigma_0 = \{q_1\} \times \mathbb{Z}^2$ such that $\alpha_{\text{CONST}}(\Sigma_0) = \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\perp, \perp) \rangle, \langle q_3, (\perp, \perp) \rangle, \langle q_4, (\perp, \perp) \rangle \rangle$ and for the valid property $P^\# = \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, \top) \rangle, \langle q_4, (\top, \top) \rangle \rangle \in \text{CONST}_2^{|Q|}$. It is easy to check that Algorithm 1 computes the following sequence $I^{k+1} = \alpha_{\text{CONST}}(\widetilde{\text{pre}}_{\mathcal{P}}(\gamma_{\text{CONST}}(I^k))) \sqcap_{\text{CONST}} I^k \sqcap_{\text{CONST}} P^\#$ in $\text{CONST}_2^{|Q|}$:

$$\begin{aligned}
I^0 &= \dot{\alpha}_{\text{CONST}}(\Sigma) = \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, \top) \rangle, \langle q_3, (\top, \top) \rangle, \langle q_4, (\top, \top) \rangle \rangle \\
I^1 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, \top) \rangle, \langle q_4, (\top, \top) \rangle \rangle \\
I^2 &= \langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, 1) \rangle, \langle q_4, (\top, \top) \rangle \rangle = \\
&= \dot{\alpha}_{\text{CONST}}(\widetilde{\text{pre}}_{\mathcal{P}}(\dot{\gamma}_{\text{CONST}}(I^3))) \dot{\geq}_{\text{CONST}} \dot{\alpha}_{\text{CONST}}(\Sigma_0)
\end{aligned}$$

The computation of I^2 derives from $\widetilde{\text{pre}}(\{q_2\} \times \mathbb{Z} \times \{2\} \cup \{q_1, q_3, q_4\} \times \mathbb{Z}^2) = \{q_1, q_2, q_4\} \times \mathbb{Z}^2 \cup \{q_3\} \times \mathbb{Z} \times \{1\}$, so $\gamma_{\text{CONST}}(I^1) \cap \gamma_{\text{CONST}}(\alpha_{\text{CONST}}(\{q_1, q_2, q_4\} \times \mathbb{Z}^2 \cup \{q_3\} \times \mathbb{Z} \times \{1\})) = \{q_1, q_4\} \times \mathbb{Z}^2 \cup \{q_2\} \times \mathbb{Z} \times \{2\} \cup \{q_3\} \times \mathbb{Z} \times \{1\}$. Thus, I^2 is returned by Alg. 1 and is the greatest inductive invariant in CONST which proves $P^\#$. In particular, the abstract inductive invariant I^2 is strictly weaker than the inductive invariant $\langle \langle q_1, (\top, \top) \rangle, \langle q_2, (\top, 2) \rangle, \langle q_3, (\top, 1) \rangle, \langle q_4, (\top, 2) \rangle \rangle$ computed in Example 4.2 for proving the same property $P^\#$ by the algorithm AINV of Corollary 3.5. \square

5.2 Backward and Forward Algorithms

We have that Algorithm 1 is backward because it applies $\widetilde{\text{pre}}$, for termination it requires that the abstract domain $\langle \hat{\mu}_L, \subseteq \rangle$ is DCC and turns out to be the dual of the forward algorithm AINV provided by Corollary 3.5 for post

and requiring that $\langle \check{\mu}_L, \subseteq \rangle$ is ACC. This latter assumption would be satisfied by dualizing the wqo \sqsubseteq_L , *i.e.*, by requiring that $\langle \Sigma, \sqsupseteq_L \rangle$ is a wqo.

A different gfp-based *forward* algorithm based can be designed by observing (as in [7, Section 3]) that $\text{lfp}(\lambda X. \Sigma_0 \cup \text{post}(X)) \subseteq P$ iff $\text{lfp}(\lambda X. \neg P \cup \text{pre}(X)) \subseteq \neg \Sigma_0$. Here, the dualization provided by the equivalence (7) yields:

$$\text{lfp}(\lambda X. \neg P \cup \check{\mu}_L(\text{pre}(X))) \subseteq \neg \Sigma_0 \Leftrightarrow \neg P \subseteq \text{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\text{post}}(X) \cap X \cap \neg \Sigma_0)).$$

This induces the following *co-inductive forward algorithm* which relies on the state transformer $\widetilde{\text{post}}$ and is terminating when $\langle \check{\mu}_L, \subseteq \rangle$ is assumed to be DCC:

Algorithm 4: Co-inductive forward abstract inductive invariant synthesis.

```

I := Σ;
while (P ⊆ I) do // Loop invariant: I ∈ L
  if (I = μL(post(I)) ∩ I ∩ ¬Σ0) then return I is an inductive invariant in L;
  I := I ∩ μL(post(I)) ∩ I ∩ ¬Σ0;
return no inductive invariant in L;

```

Furthermore, by dualizing the technique and the algorithm described in [7, Section 4.3] for *post* and *pre*, one could also design a more efficient combined forward/backward synthesis algorithm which simultaneously make backward, by $\widetilde{\text{pre}}$, and forward, by $\widetilde{\text{post}}$, steps.

6 Future Work

As hinted by Monniaux [28], results of undecidability to the question (3) for some abstract domain A display a foundational trait since they “vindicate” (often years of intense) research on precise and efficient algorithms for *approximate* static program analysis on A . To the best of our knowledge, few undecidability results are available: an undecidability result by Monniaux [28, Theorem 1] for the abstraction of convex polyhedra [8] and by Fijalkow et al. [13, Theorem 1] for semilinear sets, *i.e.* finite unions of convex polyhedra. However, convex polyhedra and semilinear sets cannot be defined by a Galois connection and therefore do not satisfy our Assumption 3.1. As an interesting and stimulating future work we plan to investigate whether the abstract inductive invariant principle could be exploited to provide a reduction of the undecidability of the question (3) for abstract domains which satisfy Assumption 3.1 and, in view of the characterization of fixpoint completeness given in Section 3.3, for transfer functions which are not fixpoint complete.

We also plan to study whether complete abstractions can play a role in the decidability result by Hrushovski et al. [17] on the computation of the strongest polynomial invariant of an affine program. This hard result in [17] relies on the Zariski closure, which is continuous for affine functions. The observation here is that the Zariski closure is pointwise complete for the transfer functions of affine programs, therefore fixpoint completeness for affine programs holds, and one could investigate whether the algorithm given in [17] may be viewed as a least fixpoint computation of a best correct approximation on the Zariski abstraction.

References

1. S. Chatterjee and M. Kishinevsky. Automatic generation of inductive invariants from high-level microarchitectural models of communication fabrics. In *Proc. 22nd International Conference on Computer Aided Verification (CAV’10)*, volume 6174 of *Lecture Notes in Computer Science*, pages 321–338. Springer, 2010.
2. P. Cousot. Partial completeness of abstract fixpoint checking. In *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation (SARA’02)*, volume 1864 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, 2000.

3. P. Cousot. On fixpoint/iteration/variant induction principles for proving total correctness of programs with denotational semantics. In *Proc. 29th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'19)*, volume to appear of *Lecture Notes in Computer Science*. Springer, 2019.
4. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.
5. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79)*, pages 269–282, New York, NY, USA, 1979. ACM.
6. P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs. In D. Néel, editor, *Tools & Notions for Program Construction: an Advanced Course*, pages 75–119. Cambridge University Press, Cambridge, UK, Aug. 1982.
7. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6(1):69–95, 1999.
8. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96. ACM, 1978.
9. I. Dillig, T. Dillig, B. Li, and K. L. McMillan. Inductive invariant generation via abductive inference. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, (OOPSLA'13)*, pages 443–456. ACM, 2013.
10. Y. M. Y. Feldman, N. Immerman, M. Sagiv, and S. Shoham. Complexity and information in invariant inference. *Proc. ACM Program. Lang.*, 4(POPL), 2019.
11. Y. M. Y. Feldman, O. Padon, N. Immerman, M. Sagiv, and S. Shoham. Bounded Quantifier Instantiation for Checking Inductive Invariants. *Logical Methods in Computer Science*, Volume 15, Issue 3, 2019.
12. Y. M. Y. Feldman, J. R. Wilcox, S. Shoham, and M. Sagiv. Inferring inductive invariants from phase structures. In *Proc. 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 405–425. Springer, 2019.
13. N. Fijalkow, E. Lefauchaux, P. Ohlmann, J. Ouaknine, A. Pouly, and J. Worrell. On the Monniaux problem in abstract interpretation. In *Proc. 26th International Static Analysis Symposium (SAS'19)*, volume 11822 of *Lecture Notes in Computer Science*, pages 162–180. Springer, 2019.
14. R. W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.
15. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.
16. M. Hecht. *Flow Analysis of Computer Programs*. North Holland, 1977.
17. E. Hrushovski, J. Ouaknine, A. Pouly, and J. Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'18)*, pages 530–539. ACM, 2018.
18. A. Ivrii, A. Gurfinkel, and A. Belov. Small inductive safe invariants. In *Proc. Formal Methods in Computer-Aided Design (FMCAD'14)*, pages 115–122. IEEE, 2014.
19. M. Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976.
20. G. A. Kildall. A unified approach to global program optimization. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL'73)*, pages 194–206. ACM, 1973.
21. Z. Kincaid, J. Cyphert, J. Breck, and T. Reps. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.*, 2(POPL), 2018.
22. K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16), Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010.
23. H. Ma, A. Goel, J. Jeannin, M. Kapritsos, B. Kasikci, and K. A. Sakallah. Towards automatic inference of inductive invariants. In *Proceedings of the Workshop on Hot Topics in Operating Systems, (HotOS'19)*, pages 30–36. ACM, 2019.
24. Z. Manna, S. Nes, and J. Vuillemin. Inductive methods for proving properties of programs. *Commun. ACM*, 16(8):491–502, 1973.
25. A. Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017.
26. A. Miné, J. Breck, and T. W. Reps. An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs. In *Proc. 25th European Symposium on Programming (ESOP'16)*, volume 9632 of *Lecture Notes in Computer Science*, pages 560–588. Springer, 2016.

27. D. Monniaux. On the decidability of the existence of polyhedral invariants in transition systems. *arXiv CoRR*, [abs/1709.04382](https://arxiv.org/abs/1709.04382), 2017.
28. D. Monniaux. On the decidability of the existence of polyhedral invariants in transition systems. *Acta Inf.*, 56(4):385–389, 2019.
29. C. Müller, H. Seidl, and E. Zalinescu. Inductive invariants for noninterference in multi-agent workflows. In *Proc. 31st IEEE Computer Security Foundations Symposium (CSF’18)*, pages 247–261. IEEE Computer Society, 2018.
30. M. Müller-Olm and H. Seidl. A note on Karr’s algorithm. In *Proceedings 31st International Colloquium on Automata, Languages and Programming (ICALP’04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1016–1028. Springer, 2004.
31. P. Naur. Proof of algorithms by general snapshots. *BIT Numerical Mathematics*, 6(4):310–316, 1966.
32. O. Padon, N. Immerman, S. Shoham, A. Karbyshev, and M. Sagiv. Decidability of inferring inductive invariants. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’16)*, pages 217–231. ACM, 2016.
33. D. Park. Fixpoint induction and proofs of program properties. *Machine Intelligence*, 5, 1969.
34. D. Park. On the semantics of fair parallelism. In D. Bjørner, editor, *Abstract Software Specifications*, pages 504–526. Springer Berlin Heidelberg, 1980.
35. J. H. Reif and H. R. Lewis. Symbolic evaluation and the global value graph. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL’77, pages 104–118, New York, NY, USA, 1977. ACM.
36. X. Rival and K. Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective*. The MIT Press, 2020.
37. S. Shoham. Undecidability of inferring linear integer invariants. *arXiv CoRR*, [abs/1812.01069](https://arxiv.org/abs/1812.01069), 2018.
38. A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, University of Cambridge Mathematical Laboratory, Cambridge, UK, <http://www.turingarchive.org/browse.php/b/8>, pages 67–69, 1949.