Incompleteness, Counterexamples and Refinements in Abstract Model-Checking

Roberto Giacobazzi[1] and Elisa Quintarelli[1,2]

[1] Dip. di Informatica, Università di Verona.
Strada Le Grazie 15, 37134 Verona (Italy).
[2] Dip. di Elettronica e Informazione, Politecnico di Milano.
Piazza Leonardo da Vinci, 20133 Milano (Italy).
E-mail: giaco|quintare@sci.univr.it

Abstract. In this paper we study the relation between the lack of completeness in abstract interpretation of model-checking and the structure of the counterexamples produced by a model-checker. We consider two dual forms of completeness of an abstract interpretation: Forward and backward completeness. They correspond respectively to the standard γ/α completeness of an abstract interpretation and can be related with each other by adjunction. We give a constructive characterization of Clarke et al.'s spurious counterexamples in terms of both forward and backward completeness of the underlying abstract interpretation. This result allows us to understand the structure of the counterexamples that can be removed by systematically refining abstract domains to achieve completeness with respect to a given operation. We apply our result to improve static program analysis by refining the model-checking of an abstract interpretation.

Keywords: Completeness, Model-checking, Verification, Abstract Interpretation, Domain Refinement, Program Analysis.

1 Introduction

Many authors recognized in the possibility of modifying abstract models by modifying abstractions a great potential for improving abstract model-checking in precision and reducing complexity (e.g. see Section 9 in [10]), but few applications of these techniques are known in the field of model-checking. In this paper we observe that there exists a strong connection between the standard notion of complete abstract interpretation [6, 7, 14] and the corresponding one for abstract model-checking [10, 2], and we show how the latter one can be achieved by minimally modifying abstract domains.

Completeness in abstract interpretation corresponds to require that no loss of precision is introduced by approximating a semantic function computed on abstract objects with respect to approximating the same computation on concrete objects. Therefore, no loss of precision is accumulated in abstract computations by approximating concrete input objects. This property comes directly from the original definition of Galois-connection based abstract interpretations [6, 7]: If

α and γ is a pair of adjoint functions in a Galois connection specifying an abstract interpretation, and $\mathcal{O}$ and $\mathcal{O}^\sharp$ are respectively the concrete and abstract operations, we say that the abstract interpretation is complete if $\alpha \circ \mathcal{O} = \mathcal{O}^\sharp \circ \alpha$. Recently, Giacobazzi et al. [14] observed that completeness for an abstract interpretation, i.e., abstract domains and abstract operations, only depends upon the structure of the underlying abstract domains and, in particular, that it is always possible to make any abstract domain A complete with respect to a concrete function $\mathcal{O}$ by minimally extending or even reducing A. This process is known as the *complete shell/core* of A. This means that a systematic domain transformer can be introduced to improve precision of domains with respect to a given function by letting the new domains be complete with respect to this function.

The problem

The idea of abstract (state-)model-checking is that of verifying temporal properties against an approximated model which is systematically derived from the concrete semantics of the system we want to analyze. This is always achieved by approximating the information contained in its states. Since the pioneering work on model-checking and abstraction by Clarke et al. [3], a number of works have applied this idea to reduce the phenomenon of "state explosion" (e.g. [8, 10, 17]). Indeed, verifying a temporal logic formula against a model is in general a hard problem (PSPACE complete for CTL*) [1, 11, 16]. The problem of refining abstract model checking is precisely that of improving precision in temporal logic verification by refining a state partition (called in [3] an abstraction) that turns out to provide a too rough abstract model for verifying a given temporal property of interest. Roughly speaking, by refining the abstraction in abstract model-checking we gain precision, namely the refined model becomes more selective (being closer to the concrete model) than the abstract one. Moreover, the size of the new refined abstract model has to be kept as small as possible in order to avoid to re-introduce the "state explosion" phenomenon. Formulated in this way, this is indeed precisely a domain refinement problem: Improve abstractions (viz. enhance domains) in such a way that the new abstraction contains the least amount of information that is needed to achieve a given precision degree [12, 13]. It is not an easy task to apply the standard abstract domain refinement theory ([7, 12, 13]) to abstract model-checking refinement. This because of two reasons: (1) Even though any state partition can be associated with a Galois connection, the reverse does not hold in general. Therefore it is not always immediate to associate a refined state partition with a refined abstract domain (or equivalently Galois connection) as the latter may not correspond to a state partition. (2) We need to express the meaning of improving precision in abstract model-checking by refining domains. This has to be related with the structure of the formulas that are verified in the new refined model. We solve both these problems in the context of abstract domain refinements to achieve complete abstract model checking.

Main results

In abstract model-checking soundness means that when a property is true in the abstract model it will also be true in the concrete one. However, the approximation may not be complete in the sense of model-checking, i.e., if the property of interest is false in the abstract model, the counterexample produced by the model-checker may be the result of some particular traces in the approximated model, which are not present in the concrete one, i.e., which do not correspond to any concrete computation. In [2] this phenomenon has been isolated, and these counterexamples have been called *spurious*. We consider spurious counterexamples as a measure for the achieved precision in systematically refining abstract model-checking by standard domain transformers. We first introduce the notions of *backward* and *forward completeness* for an abstract interpretation. While backward completeness corresponds precisely to the standard notion of completeness for an abstract interpretation [7, 14], forward completeness is less well-known and it corresponds to what is sometimes called "exactness" of an abstract interpretation (e.g. [4]). We show that forward and backward completeness are dual notions, and that, as well as for backward completeness, also forward completeness can be achieved by minimally refining abstract domains. Then we observe that any abstract domain refinement induces a refined state partition which in turn reduces the amount of spurious counterexamples in the refined abstract model. Finally we show that the counterexample-guided abstraction refinement by Clarke et al. [2] corresponds precisely to iteratively compute the backward (forward)-complete shell of the corresponding abstraction with respect to the forward (backward) state transition function. This provides a purely algebraic characterization of spurious counterexamples in terms of standard abstract interpretation theory. Moreover, this result allows us to characterize the structure of spurious counterexamples that can be removed by refining a domain to achieve completeness with respect to an arbitrary function, therefore including most domain refinements known in the literature. Our results are applied in the systematic refinement of data-flow analysis as the model-checking of abstract interpretations, as formulated in [20].

Related works

The body of research on the connection among model-checking, abstract interpretation and data-flow analysis is huge, even though this field still represents a major challenge in formal methods. In particular the problem of completeness in abstract model-checking has been studied by a number of authors both theoretically and from the point of view of improving model-checking algorithms. In [2] the authors proposed an automatic refinement technique which uses information obtained from erroneous counterexamples produced by a model-checker. They introduced the notion of spurious counterexample and gave an algorithm based on the inverse image of the forward transition function to refine the abstraction. However, the authors did not consider the classical notions of abstract interpretation in order to improve the precision of the approximated models.

In our work we prove that this algorithm can be formulated as an instance of the more general problem of making an abstraction complete in the standard sense of abstract interpretation. In [9] Cousot and Cousot introduced a novel general temporal language, inspired by Kozen's μ-calculus, which includes most standard specification languages such as CTL and CTL* as special cases. The authors proved that the classical state-based model-checking of this language is an abstract interpretation of its trace-based semantics, which turns out to be incomplete. More recently Ranzato [19] found the complete shell according to [14] of Cousot's state-based model-checking abstraction. This construction has been applied to the whole specification language introduced in [9], proving that there exists no complete abstraction which includes the state-model-checking and it is strictly more abstract than the trace-based model-checking. This proves that it is impossible for state-model-checking to achieve completeness with respect to trace-model-checking, without including the traces them self. In [5] the authors introduced a new concept of completeness, called *partial completeness*, for problems which are intended to check abstract fix-points for specifications. A checking algorithm is partially complete when in case of termination the answer is exact. This notion can be directly applied in model-checking problems, where one is interested in partially complete abstractions, i.e., approximations of a given model which always yield an affirmative answer when the specification is correct (i.e. it holds in the original model) and the algorithm does terminate. This notion of completeness seems different from the one considered in this paper, but further research is needed to better understand the relation between these two notions of completeness.

Structure of the paper

The paper is organized as follows: In Sections 2 and 3 we recall first the main notions concerning temporal logic and model checking, and then the standard theory of Galois-connection based abstract interpretation. The main results of the paper are in Section 4, Section 5, and Section 6. In Section 4 we introduce forward and backward completeness of an abstract interpretation and we study the relation between these two notions and a constructive method to minimally modify domains in order to achieve either forward or backward completeness. In Section 5 we introduce abstract model checking refinement by domain transformers and in Section 6 we characterize the precision of a refined abstract model in terms of Clarke's spurious counterexamples. Finally, in Section 7 we give a model-checking perspective of the problem of complete data-flow analysis. Section 8 concludes with future works.

2 Temporal logic and model-checking

In the following we consider a fragment $\forall$CTL* of the branching time temporal logic CTL* [3, 11]: The formulas of $\forall$CTL* do not contain existential quantifiers. Of course, all the results apply to the universal fragment of the weaker language CTL, as well. In $\forall$CTL* universal properties are expressed through the path quantifier $\forall$ ("for all futures") that quantifies over (infinite) execution sequences. The temporal operators G (**G**enerally, always), F (**F**inally, sometime) X (ne**X**t time), and U (**U**ntil) express properties of a single execution sequence. These operators, as well as other syntactic possibilities, can be freely nested in a formula. Given a set *Prop* of propositions, the set *Lit* of *literals* is defined as $\mathit{Lit} = \mathit{Prop} \cup \{\neg q \mid q \in \mathit{Prop}\} \cup \{\mathsf{true}, \mathsf{false}\}$. *State formulas* ϕ and *Path formulas* ψ are inductively defined by the following grammar, where $p \in \mathit{Lit}$:

$$\begin{array}{ll} \text{State formulas:} & \phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall \psi \\ \text{Path formulas:} & \psi ::= \phi \mid \psi \wedge \psi \mid \psi \vee \psi \mid G\psi \mid F\psi \mid X\psi \mid U(\psi, \psi) \end{array}$$

Observe that negation is allowed only for literals. Therefore, the results that allow us to write the existential modal operator using universal formulas do not hold here. Observe also that F is superfluous because $F\psi = U(\mathsf{true}, \psi)$.
A *transition system* is a pair $\langle \Sigma, R \rangle$ consisting of a set Σ of *states* and a *transition relation* $R \subseteq \Sigma \times \Sigma$ that is assumed to be total. A *Kripke structure* is a tuple $\mathbb{M} = \langle \Sigma, R, I, \|\cdot\| \rangle$ where $\langle \Sigma, R \rangle$ is a transition system, $I \subseteq \Sigma$ is the set of *initial states*, and $\|\cdot\|$: $\mathit{Lit} \longrightarrow \wp(\Sigma)$ is the *interpretation function* such that $\| p \| = \{ s \in \Sigma \mid s \models p \}$. For $\forall$CTL* the notion of *satisfaction* of a state formula ϕ by a state s ($s \models \phi$) is as usual [10]. If $\mathbb{M} = \langle \Sigma, R, I, \|\circ\| \rangle$ is a Kripke structure, we say that $\mathbb{M} \models \varphi$ if and only if $\forall s \in I : s \models \varphi$. Given a temporal formula φ the satisfiability problem for φ is that of checking if there exists a Kripke structure $\mathbb{M}$ such that $\mathbb{M} \models \varphi$. In the case of CTL* (hence of $\forall$CTL*) this problem is decidable [11]. For verification purposes the following problem is known as the *(global) model-checking problem (MCP)*: given $\mathbb{M} = \langle \Sigma, R, I, \|\circ\| \rangle$ and a formula φ, check if $\mathbb{M} \models \varphi$.

3 Abstract Interpretation

In the following $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ denotes a complete lattice C, with ordering $\leq$, *lub* $\vee$, *glb* $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$. The downward closure of $S \subseteq C$ is defined as $\downarrow S \stackrel{\text{def}}{=} \{x \in C \mid \exists y \in S.\, x \leq y\}$. $\downarrow x$ is a shorthand for $\downarrow \{x\}$, while the upward closure $\uparrow$ is dually defined. The notation $C \cong D$ denotes that C and D are isomorphic possibly ordered structures. Recall that a function $f \in C \to D$ is (Scott-)continuous if f preserves *lub*'s of (nonempty) chains iff f preserves *lub*'s of directed subsets. In the following we consider Galois connection based abstract interpretation [6]. A pair of functions $f : A \to B$ and $g : B \to A$ forms an *adjunction* if

$$\forall x \in A. \forall y \in B.\ f(x) \leq_B y \Leftrightarrow x \leq_A g(y).$$

f (resp. g) is the *left-* (*right-*)*adjoint* to g (f) and it is an additive (co-additive) function, i.e., f preserves *lub*'s (*glb*'s) of all subsets of A (empty-set included). Additive and co-additive functions f admit respectively right and left adjoint: $f^+ \stackrel{\text{def}}{=} \lambda x.\ \bigvee \{ y \mid f(y) \leq x \}$ and $f^- \stackrel{\text{def}}{=} \lambda x.\ \bigwedge \{ y \mid x \leq f(y) \}$ respectively.

Remember that $(f^+)^- = (f^-)^+ = f$. A *Galois connection* (GC for short) is an adjunction between posets, and it is denoted (A, f, B, f^+). In GC-based abstract interpretation the concrete and abstract domains, C and A, are often assumed to be complete lattices and are related by abstraction $\alpha : C \to A$ and concretization $\gamma : A \to C$ maps forming a GC (C, α, A, γ). If in addition $\forall a \in A.\ \alpha(\gamma(a)) = a$, then we call (C, α, A, γ) a Galois insertion (GI) of A in C. When (C, α, A, γ) is a GI then each value of the abstract domain A is useful in representing C, because all the elements of A represent distinct members of C, being γ 1-1. Any GC may be lifted to a GI by identifying in an equivalence class those values of the abstract domain with the same concretization. This process is known as *reduction* of the abstract domain. An *upper closure operator* on a poset C is an operator $\rho : C \to C$ which is monotone, idempotent, and extensive $(\forall x \in C.\ x \leq \rho(x))$. Lower closures are dually defined. The set of all upper (lower) closure operators on C is denoted by $uco(C)$ $(lco(C))$. Each closure is uniquely determined by the set of its fix-points $\rho(C)$. A fundamental property of closure operators is that if C is a complete lattice then both $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \lambda x.x\rangle$ ordered point-wise, and $\langle \rho(C), \leq, \vee_\rho, \wedge, \top, \rho(\bot)\rangle$ with $\vee_\rho X = \rho(\vee X)$, are complete lattices. It is well known since [7] that abstract domains can be equivalently specified either as Galois insertions or as (sets of fix-points of) upper closures on the concrete domain. In particular $X \subseteq C$ is the set of fix-points of an upper closure ρ on C iff X is a *Moore-family* of C, i.e., $X = \mathcal{M}(X) \stackrel{\text{def}}{=} \{\wedge S \mid S \subseteq X\}$ — where $\wedge\varnothing = \top \in \mathcal{M}(X)$, iff X is isomorphic to an abstract domain A in a GI (C, α, A, γ), i.e. $A \cong \rho(C)$ with $\iota : \rho(C) \to A$ and $\iota^{-1} : A \to \rho(C)$ being the isomorphism, and $(C, \iota\circ\rho, A, \iota^{-1})$ is the GI, i.e. $\rho = \gamma\circ\alpha$. Therefore $uco(C)$ is isomorphic to the so called *lattice of abstract interpretations of* C [7]. For any $X \subseteq C$, $\mathcal{M}(X)$ is called the *Moore-closure* of X in C, i.e., $\mathcal{M}(X)$ is the least (w.r.t. set-inclusion) subset of C which contains X and it is a Moore-family of C. In this case $A \sqsubseteq B$ iff $B \subseteq A$ as Moore families of C, iff A is more concrete than B. In the following we will find particularly convenient to identify closure operators (and therefore abstract domains) with their sets of fix-points, possibly by using as notation capital Latin letters.

4 Backward and forward completeness

In abstract interpretation there are two equivalent ways to express the soundness of an abstraction [6]. Let C be a complete lattice, $f : C \to C$, (C, α, A, γ) be a Galois insertion, and $f^\sharp : A \to A$. Then (C, α, A, γ) and $f^\sharp$ provide a sound abstraction of f if $\alpha\circ f \leq f^\sharp\circ\alpha$, or equivalently (by adjunction) if $f\circ\gamma \leq \gamma\circ f^\sharp$. While these two definitions of soundness are equivalent, they are not equivalent when equality is required. In the first case $\alpha\circ f = f^\sharp\circ\alpha$ means that no loss of precision is accumulated by approximating the input arguments of a given semantic function; while $f\circ\gamma = \gamma\circ f^\sharp$ means that no loss of precision is accumulated by approximating the result of computations on abstract objects. These two notions represent therefore two different forms of completeness: The first is related with input arguments of a computation, while the second is related with its output.

We distinguish here between these two forms of completeness: The first is called *backward* ($\mathcal{B}$) and the second is called *forward* ($\mathcal{F}$) completeness. The reason for these names will be clear later in the paper.

Definition 1. *Let C be a complete lattice, $f : C \to C$, (C, α, A, γ) be a Galois insertion, and $f^\sharp : A \to A$.*

- *(C, α, A, γ) and $f^\sharp$ are $\mathcal{B}$-complete for f if $\alpha \circ f = f^\sharp \circ \alpha$.*
- *(C, α, A, γ) and $f^\sharp$ are $\mathcal{F}$-complete for f if $f \circ \gamma = \gamma \circ f^\sharp$.*

$\mathcal{B}$-completeness (see [14]) corresponds to ask that the approximate function $f^\sharp$ perfectly mimics the concrete function f, when the latter is approximated in A, viz. both are compared in the abstract domain A. Conversely, $\mathcal{F}$-completeness corresponds to ask that $f^\sharp$ perfectly mimics the function f applied to the same abstract arguments, viz. they are both compared in the concrete domain C. The following proposition extends a result in [14] about $\mathcal{B}$-completeness to $\mathcal{F}$-completeness. In this case there exists an either $\mathcal{B}$ or $\mathcal{F}$ complete abstract function $f^\sharp$ in an abstract domain A iff the best correct approximation $\alpha \circ f \circ \gamma$ of f is respectively either $\mathcal{B}$ or $\mathcal{F}$ complete. This means that, as well as for $\mathcal{B}$-completeness [14], also $\mathcal{F}$-completeness is a property of abstract domains, namely a property of the GI (C, α, A, γ).

Proposition 1. *Let C be a complete lattice, $f : C \to C$, and (C, α, A, γ) be a Galois insertion. There exists $f^\sharp : A \to A$ such that (C, α, A, γ) and $f^\sharp$ are $\mathcal{B}$-complete ($\mathcal{F}$-complete) for f iff $\alpha \circ f = \alpha \circ f \circ \gamma \circ \alpha$ ($f \circ \gamma = \gamma \circ \alpha \circ f \circ \gamma$).*

Therefore, we can define when a domain (closure operator) is $\mathcal{B}/\mathcal{F}$-complete for a semantic function as follows.

Definition 2. *Let C be a complete lattice and $\rho \in uco(C)$ be an abstract domain. ρ is $\mathcal{B}$($\mathcal{F}$)-complete for f if $\rho \circ f = \rho \circ f \circ \rho$ ($f \circ \rho = \rho \circ f \circ \rho$).*

While $\mathcal{B}$-completeness is well known in abstract interpretation and it corresponds to the standard notion of completeness [18, 14], the notion of forward completeness is less known. $\mathcal{B}$-completeness means that the domain ρ is expressive enough such that no loss of precision is accumulated by abstracting in ρ the arguments of a function f. Conversely, $\mathcal{F}$-completeness means that no loss of precision is accumulated by approximating the result of the function f computed in ρ. This justifies the choice of the backward and forward names above. In the following we denote by $\mathcal{F}(C, f) \stackrel{\text{def}}{=} \left\{ \rho \,\middle|\, f \circ \rho = \rho \circ f \circ \rho \right\}$ and $\mathcal{B}(C, f) \stackrel{\text{def}}{=} \left\{ \rho \,\middle|\, \rho \circ f = \rho \circ f \circ \rho \right\}$ respectively the set of $\mathcal{F}$ and $\mathcal{B}$ complete abstractions of C for f. It is worth noting that in general $\mathcal{F}(C, f) \not\subseteq \mathcal{B}(C, f)$ and $\mathcal{F}(C, f) \not\subseteq \mathcal{B}(C, f)$, namely they are incomparable notions.

Example 1. Let $Sign^+$ be the domain in Fig. 1 which is an abstraction of $\langle \wp(\mathbb{Z}), \subseteq \rangle$ for the analysis of integer variables and $sq : \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ be the square operation defined as follows: $sq(X) = \left\{ x^2 \,\middle|\, x \in X \right\}$ for $X \in \wp(\mathbb{Z})$. Let $\rho \in uco(\wp(\mathbb{Z}))$ be the closure operator associated with $Sign^+$, i.e. $\rho = \gamma \circ \alpha$, where the abstraction and concretization maps are the most obvious ones. The best correct approximation of sq in $Sign^+$ is $sq^\sharp : Sign^+ \to Sign^+$ such that $sq^\sharp(X) = \rho(sq(X))$, with $X \in Sign^+$. It is easy to see that the upper-closure operators $\rho_a = \{\mathbb{Z}, [0, +\infty], [0, 10]\}$ and $\rho_b = \{\mathbb{Z}, [0, 2], [0]\}$, respect the following facts: $\rho_a = \{\mathbb{Z}, [0, +\infty], [0, 10]\} \in \mathcal{F}(Sign^+, sq^\sharp)$ but $\rho_a = \{\mathbb{Z}, [0, +\infty], [0, 10]\} \notin \mathcal{B}(Sign^+, sq^\sharp)$ (for example, $\rho_a(sq^\sharp(\rho_a([0]))) = [0, +\infty]$ but $\rho_a(sq^\sharp([0])) = [0, 10]$) and $\rho_b = \{\mathbb{Z}, [0, 2], [0]\} \in \mathcal{B}(Sign^+, sq^\sharp)$ but $\rho_b = \{\mathbb{Z}, [0, 2], [0]\} \notin \mathcal{F}(Sign^+, sq^\sharp)$ (e.g. $\rho_b(sq^\sharp(\rho_b([0, 2]))) = \mathbb{Z}$ but $sq^\sharp(\rho_b([0, 2])) = [0, 10]$). The arrows in Fig. 1 show the non fix-points results of $sq^\sharp$.

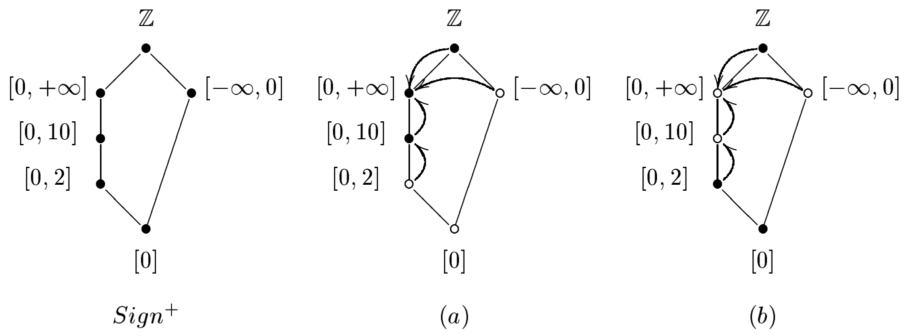


Fig. 1. The abstract domain $Sign^+$ and two abstractions

Clearly, when an abstract domain ρ is both $\mathcal{B}$ and $\mathcal{F}$ complete for f, then ρ is a morphism, i.e., $\rho \circ f = f \circ \rho$, namely $\mathcal{F}(C, f) \cap \mathcal{B}(C, f) = \{\ \rho \mid \rho \circ f = f \circ \rho\ \}$. Note that in the Example 1, $\{\mathbb{Z}, [0, +\infty], [-\infty, 0], [0]\} \in \mathcal{F}(Sign^\sharp, sq^\sharp) \cap \mathcal{B}(Sign^\sharp, sq^\sharp)$.

The following result is the main result in [14] and it is the basis for a constructive characterization of the complete shell of an abstract domain, viz. the least (most abstract) domain which is $\mathcal{B}$-complete and includes a given domain. This result constructively characterizes the structure of $\mathcal{B}$-complete abstract domains for continuous functions. Recall that, if $f : C \to C$ is a unary function, then $f^{-1}(y) = \{\ x \mid f(x) = y\ \}$.

Theorem 1 ([14]). *Let $f : C \to C$ be continuous and $\rho \in uco(C)$. Then ρ is $\mathcal{B}$-complete for f iff $\bigcup_{y \in \rho(C)} max(f^{-1}(\downarrow y)) \subseteq \rho(C)$.*

It is easy to see in Example 1 that ρ_a is not $\mathcal{B}$-complete because it does not include the maximal inverse image of $sq^\sharp$ (see Fig. 1), namely the point $[0, 2]$.

An analogous, but simpler to prove, result can be stated for $\mathcal{F}$-completeness. In this case, we can characterize $\mathcal{F}$-complete domains for merely monotone operators.

Theorem 2. *Let $f : C \to C$ be monotone and $\rho \in uco(C)$. ρ is $\mathcal{F}$-complete for f iff $\forall x \in \rho(C).\ f(x) \in \rho(C)$.*

Therefore, while $\mathcal{B}$-complete domains are closed under (maximal) inverse image of the function f, $\mathcal{F}$-complete domains are closed under direct image of f. It is easy to see in Example 1 that ρ_b is not $\mathcal{F}$-complete because it does not include the direct image of $sq^\sharp$ (see Fig. 1), namely the points $[0, 10]$ and $[0, +\infty]$. Both Theorem 1 and 2 together specify a strong relation between $\mathcal{B}$ and $\mathcal{F}$ completeness, which can be specified in terms of adjunction when f admits a right adjoint f^+.

Corollary 1. *Let C be a complete lattice and $f : C \to C$ be an additive function. Then $\mathcal{B}(\wp(S), f) = \mathcal{F}(\wp(S), f^+)$.*

Moreover it is always possible, in view of Theorem 1 and 2, to associate with each continuous semantic function $f : C \to C$ a corresponding domain refinement that transforms any abstract domain A into the closest (most abstract) $\mathcal{B}/\mathcal{F}$-complete domain for f which includes (is more concrete than) A. In the first case we obtain the ($\mathcal{B}$-)complete shell in [14], while in the second we obtain what we call the $\mathcal{F}$-complete shell.

Definition 3. *Let C be a complete lattice and $f : C \to C$ be a continuous function. We define $\mathcal{R}^{\mathcal{B}}_f : uco(C) \to uco(C)$ and $\mathcal{R}^{\mathcal{F}}_f : uco(C) \to uco(C)$ such that:*

- $\mathcal{R}^{\mathcal{B}}_f = \lambda X \in uco(C).\ \mathcal{M}(\bigcup_{y \in X} max(f^{-1}(\downarrow y)))$;
- $\mathcal{R}^{\mathcal{F}}_f = \lambda X \in uco(C).\ \mathcal{M}(f(X))$.

It is immediate to prove that both $\mathcal{R}^{\mathcal{B}}_f$ and $\mathcal{R}^{\mathcal{F}}_f$ are monotone operators on $uco(C)$. The following result, which follows by Theorem 1 and 2, characterizes in a unique domain-equational form the $\mathcal{B}/\mathcal{F}$-complete shell of abstract domains for a continuous function $f : C \to C$.

Theorem 3. *Let $f : C \to C$ be a continuous function and $A \in uco(C)$ be an abstract domain. Let $\ell \in \{\mathcal{B}, \mathcal{F}\}$:*

$$X \text{ is } \ell\text{-complete for } f \text{ and } X \sqsubseteq A \text{ iff } X = A \sqcap \mathcal{R}^{\ell}_f(X);$$

Therefore, the greatest (viz, most abstract) domain which includes A and which is ℓ-complete for f is $\mathcal{S}^{\ell}_f(A) = gfp(\lambda X.\ A \sqcap \mathcal{R}^{\ell}_f(X))$. This domain is called the *ℓ-complete shell* of A with respect to f. Note that, as proved for the $\mathcal{B}$-complete shell in [14], $\mathcal{R}^{\ell}_f$ above is a co-additive operator on the lattice of abstract interpretations $uco(C)$, and that $\lambda X.\ \mathcal{S}^{\ell}_f(X)$ is a monotone, idempotent, and reductive operator on $uco(C)$, i.e. $\mathcal{S}^{\ell}_f \in lco(uco(C))$.

5 Refining state partitions vs refining domains

In this section we study how to systematically refine abstract model-checking, namely we consider functions of type $\wp(\Sigma) \to \wp(\Sigma)$ with Σ being the set of states in a transition system $\langle \Sigma, R \rangle$. The following definition introduces the notion of abstract Kripke structure according to Dams et al. [10].

Definition 4. *Let* $\mathbb{M} = \langle \Sigma, I, R, \| \circ \| \rangle$ *be a Kripke Structure and* $(\wp(\Sigma), \alpha, A, \gamma)$ *be a GI. The corresponding abstract Kripke Structure* $\mathbb{M}^\sharp = \langle A, I^\sharp, R^\sharp, \| \circ \|^\sharp \rangle$ *is defined as follows:*

- $a \in I^\sharp$ *iff* $\exists d \in I.\ \alpha(\{d\}) = a$*;*
- $R^{\exists\exists} \stackrel{def}{=} \{\ (X, Y) \,|\, \exists_{x \in X} \exists_{y \in Y}.\ R(x, y)\ \}$*;*
- $R^\sharp(a_1, a_2)$ *iff* $a_2 \in \{\ \alpha(Y) \,|\, Y \in min\ \{\ Y' \,|\, R^{\exists\exists}(\gamma(a_1), Y')\ \}\ \}$*;*
- $\| p \|^\sharp \stackrel{def}{=} \{\ a \in A \,|\, \gamma(a) \subseteq \| p \|\ \}$.

Note that for any $a \in A$, the minimal sets Y' such that $R^{\exists\exists}(\gamma(a), Y')$ are all singletons.

In the following we consider abstract domains as specified by a GI, or equivalently by a closure operator on $\wp(\Sigma)$, where Σ is the set of states of a transition system $\langle \Sigma, R \rangle$. This is apparently in contrast with the original (standard) definition of abstract model-checking by Clarke et al. [3]. In this latter case the abstraction is specified by a surjection function, called *state partition*, $h : \Sigma \to S$ on a given set S of *abstract states*. Notice that any surjection $h : \Sigma \to S$ induces an equivalence relation $\equiv$ on the set of states Σ in the following way: Let $a, b \in \Sigma$, then $a \equiv b$ iff $h(a) = h(b)$. Given a surjection $h : \Sigma \to S$ we can immediately derive a GI in the following way: $(\wp(\Sigma), \alpha_h, \wp(S), \gamma_h)$, where $\alpha_h(X) \stackrel{\text{def}}{=} \{\ h(x) \,|\, x \in X\ \}$ is the abstraction function and $\gamma_h(Y) \stackrel{\text{def}}{=} \{\ x \,|\, h(x) \in Y\ \}$ is the concretization function. Note that the GI is defined by considering $\wp(S)$ instead of S, because it is not assumed that the set of abstract states S is partial ordered.

The main problem with this approach and abstract domain refinement is that, in general, if $f : \wp(\Sigma) \to \wp(\Sigma)$ is some predicate transformer on the transition system $\langle \Sigma, R \rangle$, then the refined domain $\mathcal{S}^\ell_f(\gamma_h \circ \alpha_h) \in uco(\wp(\Sigma))$, with $\ell \in \{\mathcal{F}, \mathcal{B}\}$, may not be of the form $\wp(S)$ for some set of abstract states S. In particular $\mathcal{S}^\ell_f(\gamma_h \circ \alpha_h)$ may not be a Boolean algebra, and therefore the powerset of some set. Because we abstract state-model-checking, it is sufficient to observe how the refined domain abstracts each single state. Indeed, any abstract domain refinement induces a domain surjection in the following way: Let $\widehat{S} \cong \{\mathcal{S}^\ell_f(\gamma_h \circ \alpha_h)(\{x\}) \mid x \in \Sigma\}$, then the function $\widehat{h} \stackrel{\text{def}}{=} \lambda x \in \Sigma.\ \mathcal{S}^\ell_f(\gamma_h \circ \alpha_h)(\{x\})$ is a surjection from Σ into $\widehat{S}$ such that: $|S| \leq |\widehat{S}|$ and $(\wp(\Sigma), \alpha_{\widehat{h}}, \wp(\widehat{S}), \gamma_{\widehat{h}})$ is a GI. This corresponds to improve the partition induced by h into a finer partition of the concrete states as induced by $\widehat{h}$. This improvement is obtained by considering a new set of abstract states each one consisting of the (unique) best possible abstraction of each concrete state according to the refined closure $\mathcal{S}^\ell_f(\gamma_h \circ \alpha_h)$.

Proposition 2. *Let* $h : \Sigma \to S$ *be a state partition and* $(\wp(\Sigma), \alpha_h, \wp(S), \gamma_h)$ *be the corresponding GI. If* $\mathcal{S} : uco(\wp(\Sigma)) \to uco(\wp(\Sigma))$ *is a domain refinement, i.e. a monotone function such that* $\forall X \in uco(\wp(\Sigma)).\ \mathcal{S}(X) \sqsubseteq X$*, then the function* $h_{\mathcal{S}} : \Sigma \to \{\mathcal{S}(\gamma_h \circ \alpha_h)(\{x\}) \mid x \in \Sigma\}$ *such that* $h_{\mathcal{S}} = \lambda x.\ \mathcal{S}(\gamma_h \circ \alpha_h)(\{x\})$ *is a refinement of the state partition* h.

Therefore, the standard theory of abstract domain refinement (see [13]) can be fully applied directly to refine any abstract (state-)model-checking problem. In this latter case, a refined state partition function can be systematically derived from the refined abstract domain as described in Proposition 2.

Example 2. Consider the transition system $\langle [0,9], R\rangle$ where the transition relation R is shown on the left in Fig. 2 with continuous lines. Consider the state partition function $h(x) = x \ \mathtt{mod}\ 4$. Following Definition 4, this induces an abstract Kripke structure associated with the GI $(\wp([0,9]), \alpha_h, \wp(\{a,b,c,d\}), \gamma_h)$ as depicted in Fig. 2 with empty boxes and dashed lines. Let $f : \wp([0,9]) \to \wp([0,9])$ be such that $f(X) = \{3\}$. The forward complete shell of $\gamma_h \circ \alpha_h$ is $gfp(\lambda X.\ \gamma_h \circ \alpha_h \sqcap \mathcal{R}^{\mathcal{F}}_f(X)) = \mathcal{R}^{\mathcal{F}}_f(\gamma_h \circ \alpha_h)$, and it is immediate to observe that it induces a new state partition function $\widehat{h}$ such that

$$\widehat{h}(x) = \begin{cases} x & \text{if } x = 3, 7 \\ h(x) & \text{otherwise} \end{cases}$$

The GI $(\wp([0,9]), \alpha_h, \wp(\{a,b,c,d\}), \gamma_h)$ is not $\mathcal{F}$-complete for f, e.g. if $\rho_h = \gamma_h \circ \alpha_h$ then $f(\rho_h(\{2,6\})) = \{3\}$ but $\rho_h(f(\rho_h(\{2,6\}))) = \{3,7\}$. In this case, in order to obtain the complete shell of ρ_h we have to add the abstract state $\{3\}$ to the abstract domain. This leads to a partition of the abstract state $\{3,7\}$ into two more concrete states: $\{3\}$ which is introduced by adding the forward image of f, and $\{7\}$ which is a new state (actually, it is a renaming of the state $\{3,7\}$) corresponding to the (unique) best possible approximation in $\mathcal{R}^{\mathcal{F}}_f(\gamma_h \circ \alpha_h)$ of the concrete state 7. This induces a refined abstract Kripke structure associated with the refined GI $(\wp([0,9]), \alpha_{\widehat{h}}, \wp(\{a,b,c,d,e\}), \gamma_{\widehat{h}})$, as shown on the right in Fig. 2.

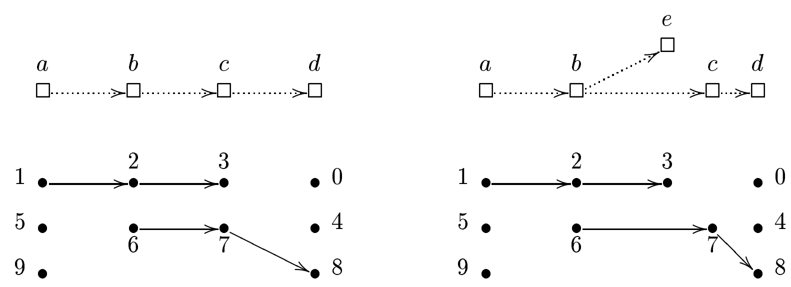


Fig. 2. An abstract Kripke structure and its refinement

6 Completeness and counterexamples

We are in the position to apply forward or backward completeness abstract domain refinement to abstract (state-)model-checking problems. In this section we relate the abstract domain refinement for both backward and forward completeness with the presence of suitable counterexamples in abstract model-checking.

A *path* in a Kripke Structure $\mathbb{M} = \langle \Sigma, I, R, \| \circ \| \rangle$ is an infinite sequence in $\mathbb{N} \to \Sigma$, denoted $\pi = s_0, s_1, \ldots$, such that $s_0 \in I$ and for every $i \in \mathbb{N}$, $R(s_i, s_{i+1})$ holds. Traditionally, a terminating execution has a final state which is repeated forever. With π_i we denote i-th state in the path, i.e., s_i. Given a Galois insertion $(\wp(\Sigma), \alpha, A, \gamma)$ and concrete and abstract paths π and $\pi^\sharp$ respectively, we denote by $\alpha(\pi)$ the sequence $\alpha(\pi_0), \alpha(\pi_1), \ldots$ and by $\gamma(\pi^\sharp)$ the set $\{ \pi \mid \alpha(\pi) = \pi^\sharp \}$. Given a path π and a state $x \in \Sigma$, $x \in \pi$ iff $\exists i \in \mathbb{N}$ such that $x = \pi_i$. The forward predicate transformer associated with R is $post[R] : \wp(\Sigma) \to \wp(\Sigma)$, which is defined as $post[R](X) = \{ s \in \Sigma \mid R(x, s) \wedge x \in X \}$. Dually, we can define the backward predicate transformer associated with R as $\widetilde{pre}[R] : \wp(\Sigma) \to \wp(\Sigma)$, which is defined as $\widetilde{pre}[R](X) = \{ s \in \Sigma \mid \forall x \in \Sigma.\ R(s, x) \Rightarrow x \in X \}$. The following is a well known fact about forward and backward predicate transformers (e.g. [15]).

Proposition 3. *$(\wp(\Sigma), post[R], \wp(\Sigma), \widetilde{pre}[R])$ is a Galois connection.*

Let us consider a domain refinement which is specified as the $\mathcal{F}/\mathcal{B}$-complete shell with respect to a continuous function $f : \wp(\Sigma) \to \wp(\Sigma)$ as given in Theorem 1. Let $R_f \subseteq \Sigma \times \Sigma$ be the relation associated with f, i.e. such that $R_f(x, y) \Leftrightarrow y \in f(\{x\})$. It is immediate to observe that $f = post[R_f]$.

Proposition 4. *Let $f : \wp(\Sigma) \to \wp(\Sigma)$ be additive. Then we have*

$$\mathcal{B}(\wp(\Sigma), f) = \mathcal{B}(\wp(\Sigma), post[R_f]) = \mathcal{F}(\wp(\Sigma), f^+) = \mathcal{F}(\wp(\Sigma), \widetilde{pre}[R_f]).$$

Traditionally, a model-checker is used to determine whether a temporal formula holds in a model. If the answer is negative, i.e., the model does not possess the property specified by the formula, then the model checker generates a counterexample. Let $\mathcal{S} : uco(C) \to uco(C)$ be an abstract domain refinement, i.e., a monotone operator such that $\mathcal{S}(X) \sqsubseteq X$ for any abstract domain $X \in uco(C)$ [13]. It is immediate to observe that $\mathcal{S}$ reduces the number of counterexamples that might be generated by the model-checker. Recall that, given a Kripke structure $\mathbb{M} = \langle \Sigma, I, R, \| \circ \| \rangle$ and a formula $\varphi \in \forall\text{CTL*}$, $\mathbb{M} \not\models \varphi$ iff $\exists s \in I.\ s \not\models \varphi$, iff there exists a path starting at I which does not satisfy φ. This because $\forall$CTL* can only express formulas that describe behaviors that should hold on *all paths* from the initial states. Hence, if A is an abstract domain and $\mathbb{M}^A$ and $\mathbb{M}^{\mathcal{S}(A)}$ are respectively the abstract and refined Kripke structures associated with A and $\mathcal{S}(A)$, then we have that:

$$\{ \varphi \mid \mathbb{M}^A \not\models \varphi \} \supseteq \{ \varphi \mid \mathbb{M}^{\mathcal{S}(A)} \not\models \varphi \}.$$

It is therefore natural to consider counterexamples as a measure of the achieved precision in abstract model-checking when the abstract domain is refined. However, not all counterexamples are admissible, because an abstract counterexample might not be valid in the concrete model. Clarke's et al. [2] introduced the notion of *spurious counterexample* in order to model this situation[1].

[1] Note that we consider path counterexamples only. Clarke's et al. [2] consider instead *loop* counterexamples, which reduce to paths by means of loop unfolding.

Definition 5. *Let $(\wp(\Sigma), \alpha, S^\sharp, \gamma)$ be a GI. Consider the concrete and abstract Kripke structures $\langle \Sigma, I, R, \| \circ \| \rangle$ and $\langle S^\sharp, I^\sharp, R^\sharp, \| \circ \|^\sharp \rangle$ and $\varphi \in \forall CTL^* \cap LTL$. An abstract path $\pi^\sharp$ is a spurious counterexample if $\pi_0^\sharp \not\models \varphi$ and $\gamma(\pi^\sharp) = \varnothing$.*

The following theorem relates spurious counterexamples with backward and forward completeness with respect to $post[R]$ and $\widetilde{pre}[R]$ respectively. The idea is that if a counterexample is spurious for an abstraction ρ, then there exists a concrete state x for which ρ fails to be forward complete for $post[R]$ in x. The proof relies upon the fact that for spurious counterexamples $\pi^\sharp$, $\gamma(\pi^\sharp) = \varnothing$ and therefore there exists a concrete state x from which there is no forward transition into an abstract state, while there is a forward transition from $\rho(\{x\})$. This clearly violates completeness for $post[R]$.

Theorem 4. *Let $(\wp(\Sigma), \alpha, S^\sharp, \gamma)$ be a GI and $\rho = \gamma \circ \alpha$. Let $\varphi \in \forall CTL^*$ and $\pi^\sharp$ be a counterexample. If $\pi^\sharp$ is spurious then there exists $i \in \mathbb{N}$ such that $\forall x \in \gamma(\pi_i^\sharp)$ reachable from the states in I, i.e., $\gamma(\pi_0^\sharp) \cap I$, it holds that:*

$$\rho(post[R](\rho(\{x\}))) \neq \rho(post[R](\{x\})).$$

In particular, by Theorem 4, if $\pi^\sharp$ is spurious then there exists $\pi \in \gamma(\pi^\sharp)$ and $x \in \pi$ such that $\rho(post[R](\rho(\{x\}))) \neq \rho(post[R](\{x\}))$, or equivalently, by Proposition 4, there exists $x \in \pi$ such that $\rho(\widetilde{pre}[R](\rho(\{x\}))) \neq \widetilde{pre}[R](\rho(\{x\}))$.

The complete shell abstract domain refinement can therefore be used to systematically refine abstract model-checking in order to eliminate spurious counterexamples. In particular, the machinery developed in [2] to eliminate spurious counterexamples in abstract Kripke structures can be formally reduced to the complete shell refinement operation introduced in [14]. The following example, taken from [2], shows this phenomenon.

Example 3. Consider a program with a variable whose values range in the following interval $D = [1, 12]$ and which is represented by a transition system with states $\Sigma = \{1, \cdots, 12\}$, initial states $I = \{1, 2, 3, 7, 8, 10, 11\}$ and transition relation:

$$R = \left\{ \begin{array}{c} \langle 1,4\rangle, \langle 2,5\rangle, \langle 3,6\rangle, \langle 4,9\rangle, \langle 5,9\rangle, \langle 6,6\rangle, \langle 7,12\rangle, \\ \langle 8,8\rangle, \langle 9,9\rangle, \langle 10,10\rangle, \langle 11,11\rangle, \langle 12,12\rangle \end{array} \right\}$$

Let $S^\sharp$ be the set of four abstract states $S^\sharp = \{\hat{1}, \hat{2}, \hat{3}, \hat{4}\}$, corresponding to the equivalence classes of concrete states $\{1, 2, 3\}$, $\{4, 5, 6\}$, $\{7, 8, 9\}$, and $\{10, 11, 12\}$. Suppose that a model-checker gives an abstract counterexample $\pi^\sharp = \hat{1}, \hat{2}, \hat{3}, \hat{4}$. It is easy to see that $\pi^\sharp$ is spurious, in fact, this abstract trace does not correspond to any concrete trace in the transition system. By Theorem 4 we can note that $\gamma(\hat{3}) = \{7, 8, 9\}$, moreover only the concrete state 9 of this set is reachable from initial states in I, but it does not have any transition into a different concrete state. Therefore 9 does not have a transition to a state which is abstracted by $\hat{4}$.

Note that, if there exists $\pi \in \gamma(\pi^\sharp)$ and $x \in \pi$ such that $\rho(post[R](\{x\})) \neq \rho(post[R](\rho(\{x\})))$ then this may not imply that $\pi^\sharp$ is spurious. Consider for instance the abstract path $\pi^\sharp = \widehat{1}, \widehat{2}, \widehat{3}$ of Example 3. It is easy to check that $\rho(post[R](\rho(\{6\}))) = [4,9]$, whereas $\rho(post[R](\{6\})) = [4,6]$, but $\pi^\sharp$ corresponds to the concrete paths $1,4,9$ and $1,5,9$. Therefore, the complete shell refinement may induce a too strong refinement in the domain with respect to what is strictly necessary to remove spurious counterexamples, as shown in the following example. This phenomenon can be controlled by checking, at each iterate in the *gfp*-computation of the complete shell, the presence of spurious counterexamples.

Example 4. Consider the transition system in Example 3. Let $(\wp([1,12]), \alpha, A, \gamma)$ be the corresponding GI, with $A = \wp(\{[1,3],[4,6],[7,9],[10,12]\})$. By Theorem 1 and Proposition 4, the $\mathcal{B}$-complete shell $gfp(\lambda X.\ \gamma\circ\alpha \sqcap \mathcal{R}^{\mathcal{B}}_{post[R]}(X))$ is obtained as the limit of the sequence of abstract domains $\{A^i\}_{i\in\mathbb{N}}$ defined as follows:

$$A^i = \begin{cases} A & \text{if } i = 0 \\ \mathcal{M}(\widetilde{pre}[R](A^{i-1}) \cup A) & \text{otherwise} \end{cases}$$

This sequence of abstract domains converges in three steps to the fix-point. Each abstract domain A^i induces a refined partition according to Proposition 2 in the following way:

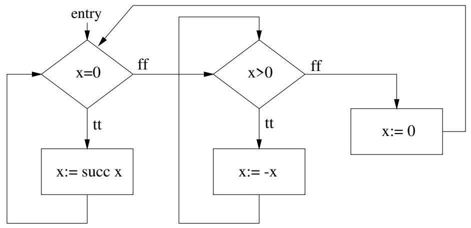
Domains		Refined partition
A^0	$\longrightarrow$	$\{[1,3],[4,6],[7,9],[10,12]\}$
A^1	$\longrightarrow$	$\{[1,3],[4,5],[6,6],[7,7],[8,9],[10,12]\}$
A^2	$\longrightarrow$	$\{[1,2],[3,3],[4,5],[6,6],[7,7],[8,9],[10,12]\}$ (fix-point)

Consider the partition induced by A^1. The new abstract states $\{4,5\}$ and $\{7\}$ are introduced by computing the maximal inverse image of $post[R]$ in the states $\{10,11,12\}$ and $\{7,8,9\}$, whereas the states $\{6\}$ and $\{8,9\}$ are added to obtain a partition of $\wp([1,12])$. Now consider the partition induced by A^2. The new abstract states $\{3\}$ and $\{1,2\}$ are introduced by computing respectively the inverse image of $post[R]$ in the states $\{6\}$ and $\{4,5\}$ of A^1.
It is worth noting that the partition induced by A^1 is sufficient to remove any spurious counterexample. Therefore the refinement process can be stopped after the first iteration. This condition can be verified by checking counterexamples at each iterates in the fix-point computation.

7 Complete data-flow analysis: A different perspective

In this section we sketch, by means of a simple example, how we can design sound and complete data-flow analysis algorithms as complete model-checking of abstract interpretations. Once again, completeness is achieved by removing spurious counterexamples.

The idea of data-flow analysis as model-checking goes back to Steffen's pioneering work [22]. More recently Schmidt [20, 21] showed that data-flow analysis problems can be formulated as model-checking problems of a *trace-based abstract interpretation* of operational semantics. Intuitively, the concrete semantics of a program is a tree whose paths represent execution traces and whose nodes display the program's changing states. The abstract interpretation of the semantics of a program is also a computation tree, but whose nodes contain the abstraction of information contained in concrete states. To be useful, a program abstract computation tree must safely simulate the concrete computation tree that it represents. Thus any property that holds in an abstract tree must also hold for all the corresponding concrete trees. However, the dual notion of completeness does not hold in general, namely the properties that do not hold in the abstract may not be counterexamples for the concrete.



`Concrete semantics transitions`
`Val=Int`
$0 \vdash x = 0 \rightarrow 0 \vdash x := succ(x)$
$0 \vdash x := succ(x) \rightarrow 1 \vdash x = 0$
$+(n+1) \vdash x = 0 \rightarrow +(n+1) \vdash x > 0$
$+(n+1) \vdash x > 0 \rightarrow +(n+1) \vdash x := -x$
$+(n+1) \vdash x := -x \rightarrow -(n+1) \vdash x > 0$
$-(n+1) \vdash x = 0 \rightarrow -(n+1) \vdash x > 0$
$-(n+1) \vdash x > 0 \rightarrow -(n+1) \vdash x := 0$
$-(n+1) \vdash x := 0 \rightarrow 0 \vdash x = 0$

Fig. 3. Flowchart and concrete interpretation

Let us consider the flowchart program depicted in Fig. 3. Its concrete semantics is expressed as a set of transition rules between program states. A program state is an element of $Val \times ProgramPoint$ and is written as $n \vdash p$, where n is the value of the variable x at program point p. Note that in this example the concrete computation tree is an infinite execution trace.
The fundamental idea in Schmidt's trace-based abstract interpretation [20, 21] is to replace an execution trace of the program by transition rules that use abstract values instead of run-time values. The result is an *abstract computation tree*, where multiple possible executions traces are presented by a non-deterministic branching in the abstract tree.

In the example above we consider a program analysis which detects whether x is zero or not. This corresponds to replace the concrete domain $Val = Int$ by $AbsVal = \{0, \neq 0\}$ and by rewriting the state transition rules accordingly. More in detail, we consider a function $\beta : Val \rightarrow AbsVal$, which maps concrete values to their most precise abstractions. For the example, in Fig. 3, we have $\beta(0) = 0$ and $\beta(\pm(n+1)) = \neq 0$. Moreover, the following *homomorphism property* is necessary in order to let the abstract transitions correctly simulate the concrete ones: if $(v \vdash p) \rightarrow (v' \vdash p')$ is a concrete transition, then there must exist an abstract transition, $(\beta(v) \vdash p) \rightarrow (a' \vdash p')$, such that $\beta(v') \leq a'$, where $\leq$ is the approximation order between abstract values. The resulting abstract semantics transitions are shown in Fig. 4:

`Abstract semantics transitions`	`Refined Abstract semantics transitions`
$0 \vdash x = 0 \to 0 \vdash x := succ(x)$	$0 \vdash x = 0 \to 0 \vdash x := succ(x)$
$0 \vdash x := succ(x) \to \neq 0 \vdash x = 0$	$0 \vdash x := succ(x) \to + \vdash x = 0$
$\neq 0 \vdash x = 0 \to \neq 0 \vdash x > 0$	$+ \vdash x = 0 \to + \vdash x > 0$
$\neq 0 \vdash x > 0 \to \neq 0 \vdash x := -x$	$+ \vdash x > 0 \to + \vdash x := -x$
$\neq 0 \vdash x > 0 \to \neq 0 \vdash x := 0$	$+ \vdash x := -x \to - \vdash x > 0$
$\neq 0 \vdash x := -x \to \neq 0 \vdash x > 0$	$- \vdash x = 0 \to - \vdash x > 0$
$\neq 0 \vdash x := 0 \to 0 \vdash x = 0$	$- \vdash x > 0 \to - \vdash x := 0$
	$- \vdash x := 0 \to 0 \vdash x := 0$

Fig. 4. Abstract Interpretation of flowchart

Consider the safety CTL formula $\varphi = \forall G \forall F(x = 0)$ specifying that *for each program state in any trace it is possible to reach a program state where the value of the variable x is* 0. It is easy to check that the abstract tree does not satisfy this property because there exists an infinite trace $\neq 0 \vdash x = 0 \to \neq 0 \vdash x > 0 \to \neq 0 \vdash x := -x \to \neq 0 \vdash x > 0 \to \ldots$ that invalidates the specification. In this case, the abstraction is too rough because this is an infinite abstract trace that does not correspond to any concrete trace, i.e., we have found a spurious counterexample for φ.

In order to find a sound and complete trace based abstract interpretation, we have to refine the abstraction. By applying the backward complete shell refinement for $post[R]$, with R being the transition relation specified above, we obtain a refined model which includes the abstract values $\{0, +, -, \neq 0\}$. In this case, the abstraction function is such that $\beta(0) = 0$, $\beta(+(n+1)) = +$ and $\beta(-(n+1)) = -$. The resulting refined abstract semantics transitions, according to Definition 4, are shown in Fig. 4. It is easy to check that the refined abstract tree now satisfies the property φ.

8 Conclusions

In this work we have studied the impact of the notions of forward and backward completeness in abstract interpretation based model-checking. Our results allows us to apply standard abstract interpretation theory to refine abstract models when they result too rough for verifying temporal properties of interest. We have proved that the precision of the abstract model-checking problem for a Kripke structure $\mathbb{M} = \langle \Sigma, I, R, \| \circ \| \rangle$ is strictly connected to the precision of the underlying abstract domain, and that spurious counterexamples can be removed by considering the complete shell of the abstract domain with respect to the $post[R]$ predicate transformer.

It is particularly interesting to study what kind of counterexamples that can be removed when the refinement is computed with respect to an arbitrary function f, possibly different from $post[R]$. In this case our results allows us to formally derive the structure of those counterexamples that can be removed by an arbitrary domain refinement operation, provided that this operation can be encoded as a complete shell, namely it corresponds to the complete shell refinement $\mathcal{S}_f^{\mathcal{B}}$ with respect to a given function f. It is easy to see that if $\mathbb{M}^A$ is an abstract Kripke structure associated with an abstract domain A and a concrete transition system $\langle \Sigma, R \rangle$, the refinement operation $\mathcal{S}_f^{\mathcal{B}}$ returns a refined Kripke structure $\mathbb{M}^{\mathcal{S}_f^{\mathcal{B}}(A)}$ according to Proposition 2 which, for a given formula φ, removes all the counterexamples that corresponds to the traces in a modified (simplified) transition system $\langle \Sigma, R \wedge R_f \rangle$. This new transition system includes only those transitions that satisfy both the original relation R and the relation associated with f. The consequences of this observation in static program analysis by abstract model checking are under investigation.

Acknowledgments

The interest in applying the constructive characterization of completeness in [14] to Clarke's spurious counterexamples came to our attention in a discussion we had with Andreas Podelsky during a breakfast at Place de la Sorbonne, Paris, in July 2000. We wish to thank Andreas for having pointed to us the work of Clarke, Francesco Ranzato, and the anonymous referees for their helpful comments. Part of this work was carried out while the authors were visiting the Laboratoire d'Informatique, Ècole Polytechnique, Palaiseau, France. This work has been supported by the Italian MURST project: *"Certificazione automatica di programmi mediante interpretazione astratta"*.

References

1. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specification. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.
2. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of the 12th Internat. Conf. on Computer Aided Verification (CAV '00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, Berlin, 2000.
3. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.
4. P. Cousot. Types as abstract interpretations (Invited Paper). In *Conference Record of the 24th ACM Symp. on Principles of Programming Languages (POPL '97)*, pages 316–331. ACM Press, New York, 1997.
5. P. Cousot. Partial completeness of abstract fixpoint checking, invited paper. In *Proceedings of the Fourth International Symposium on Abstraction, Reformulations and Approximation, SARA'2000, Lecture Notes in Artificial Intelligence 1864*, pages 1–25, Horseshoe Bay, Texas, USA, 26–29 July 2000. Springer-Verlag, Berlin, Germany.

6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symp. on Principles of Programming Languages* (*POPL '77*), pages 238–252. ACM Press, New York, 1977.
7. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symp. on Principles of Programming Languages* (*POPL '79*), pages 269–282. ACM Press, New York, 1979.
8. P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6(1):69–95, 1999.
9. P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.
10. D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.
11. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, Amsterdam and The MIT Press, Cambridge, Mass., 1990.
12. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view of abstract domain design. *ACM Comput. Surv.*, 28(2):333–336, 1996.
13. R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming* (*ICALP '97*), volume 1256 of *Lecture Notes in Computer Science*, pages 771–781. Springer-Verlag, Berlin, 1997.
14. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *Journal of the ACM*, 47(2):361–416, March 2000.
15. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods Syst. Des.*, 6:1–36, 1995.
16. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, Berlin, 1992.
17. M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking. A tutorial introduction. In G. Filé, editor, *Proceedings of the International Static Analysis Symposium* (*SAS '99*), volume 1694 of *Lecture Notes in Computer Science*, pages 330–354. Springer-Verlag, Berlin, 1999.
18. A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proc. of the ACM Symp. on Partial Evaluation and Program Manipulation* (*PEPM '93*), pages 179–185. ACM Press, New York, 1993.
19. F. Ranzato. On the completeness of model checking. In *Proc. of European Symposium on Programming* (*ESOP '01*), Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2001.
20. D. A. Schmidt. Data flow analysis is model checking of abstract interpretation. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (*POPL '98*), pages 38–48. ACM Press, New York, 1998.
21. D. A. Schmidt. Trace-based abstract interpretation of operational semantics. *Lisp and Symbolic Computation*, 10(3):237–271, 1998.
22. B. Steffen. Data flow analysis as model checking. In A. Meyer, editor, *Proc. of Theoretical aspects of computer software* (*TACS '91*), volume 526 of *Lecture Notes in Computer Science*, pages 346–364. Springer-Verlag, Berlin, 1991.