Example-Guided Abstraction Simplification

Roberto Giacobazzi[1] and Francesco Ranzato[2]

[1] University of Verona, Italy
[2] University of Padova, Italy

Abstract. In static analysis, approximation is typically encoded by abstract domains, providing systematic guidelines for specifying approximate semantic functions and precision assessments. However, it may well happen that an abstract domain contains redundant information for the specific purpose of approximating a given semantic function modeling some behavior of a system. This paper introduces Example-Guided Abstraction Simplification (EGAS), a methodology for simplifying abstract domains, i.e. removing abstract values from them, in a maximal way while retaining exactly the same approximate behavior of the system under analysis. We show that, in abstract model checking and predicate abstraction, EGAS provides a simplification paradigm of the abstract state space that is guided by examples, meaning that it preserves spuriousness of examples (i.e., abstract paths). In particular, we show how EGAS can be integrated with the well-known CEGAR (CounterExample-Guided Abstraction Refinement) methodology.

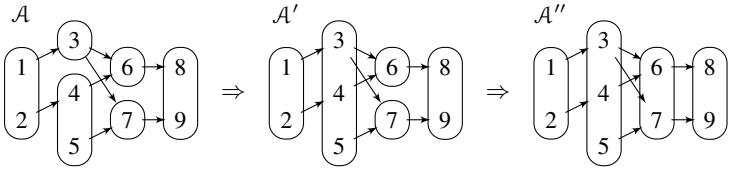
1 Introduction

In static analysis and verification, model-driven *abstraction refinement* has emerged in the last decade as a fundamental method for improving abstractions towards more precise yet efficient analyses. The basic idea is simple: given an abstraction modeling some observational behavior of the system to analyze, refine the abstraction in order to remove the artificial computations that may appear in the approximate analysis by considering how the concrete system behaves when false alarms or spurious traces are encountered. The general concept of using spurious counterexamples for refining an abstraction stems from CounterExample-Guided Abstraction Refinement (CEGAR) [4,5]. The model here drives the automatic identification of prefixes of the counterexample path that do not correspond to an actual trace in the concrete model, by isolating abstract (failure) states that need to be refined in order to eliminate that spurious counterexample. Model-driven refinements, such as CEGAR, provide algorithmic methods for achieving abstractions that are complete (i.e., precise [17,14]) with respect to some given property of the concrete model.

We investigate here the dual problem of *abstraction simplification*. Instead of refining abstractions in order to eliminate spurious traces, our goal is to simplify an abstraction A towards a simpler (ideally, the simplest) model A_s that maintains the same approximate behavior as A does. In abstract model checking, this abstraction simplification has *to keep the same examples* of the concrete system in the following sense. Recall that an abstract path π in an abstract transition system is *spurious* when no real concrete path is

S. Abramsky et al. (Eds.): ICALP 2010, Part II, LNCS 6199, pp. 211–222, 2010.

abstracted to π. Assume that a given abstract state space A of a system $\mathcal{A}$ gets simplified to A_s and thus gives rise to a more abstract system $\mathcal{A}_s$. Then, we say that $\mathcal{A}_s$ keeps the same examples of $\mathcal{A}$ when the following condition is satisfied: if π_{A_s} is a spurious path in the simplified abstract system $\mathcal{A}_s$ then there exists a spurious path π_A in the original system $\mathcal{A}$ that is abstracted to π_{A_s}. Such a methodology is called EGAS, Example-Guided Abstraction Simplification, since this abstraction simplification does not add spurious paths, namely, does keep examples, since each spurious path in $\mathcal{A}_s$ comes as an abstraction of a spurious path in $\mathcal{A}$. Let us illustrate how EGAS works through a simple example.



Let us consider the above abstract transition system $\mathcal{A}$, where concrete states are numbers which are abstracted by blocks of a state partition. The abstract state space of $\mathcal{A}$ is simplified by merging the abstract states $[3]$ and $[4,5]$: EGAS ensures that this can be safely done because $\mathrm{pre}^\sharp([3]) = \mathrm{pre}^\sharp([4,5])$ and $\mathrm{post}^\sharp([3]) = \mathrm{post}^\sharp([4,5])$, where $\mathrm{pre}^\sharp$ and $\mathrm{post}^\sharp$ denote, resp., the abstract predecessor and successor functions. This abstraction simplification leads to the above abstract system $\mathcal{A}'$. Let us observe that the abstract path $\langle [1,2],[3,4,5],[7],[8,9]\rangle$ in $\mathcal{A}'$ is spurious and it is the abstraction of the spurious path $\langle [1,2],[4,5],[7],[8,9]\rangle$ in $\mathcal{A}$. On the other hand, consider the path $\pi' = \langle [1,2],[3,4,5],[6],[8,9]\rangle$ in $\mathcal{A}'$ and observe that all the paths in $\mathcal{A}$ that are abstracted to π', i.e. $\langle [1,2],[3],[6],[8,9]\rangle$ and $\langle [1,2],[4,5],[6],[8,9]\rangle$, are not spurious. This is consistent with the fact that π' actually is not spurious. Likewise, $\mathcal{A}'$ can be further simplified to the abstract system $\mathcal{A}''$ where $[6]$ and $[7]$ are merged into a new abstract state $[6,7]$ and this transformation also keeps examples because now there is no spurious path in $\mathcal{A}''$. Let us also notice that if $\mathcal{A}$ would get simplified to $\mathcal{A}'''$ by merging $[1,2]$ and $[3]$ into a new abstract state $[1,2,3]$ then this transform would not keep examples because we would obtain the spurious loop path $\langle [1,2,3],[1,2,3],[1,2,3],...\rangle$ in $\mathcal{A}'''$ while no corresponding spurious abstract path would exist in $\mathcal{A}$.

EGAS is formalized within the standard abstract interpretation framework [8,9]. This ensures that EGAS can be applied both in abstract model checking and in abstract interpretation. Consider for instance the abstract domains $A_1 \triangleq \{\mathbb{Z}, \mathbb{Z}_{\leq 0}, \mathbb{Z}_{\geq 0}, 0\}$ and $A_2 \triangleq \{\mathbb{Z}, \mathbb{Z}_{\geq 0}\}$ for sign analysis of an integer variable. Recall that in abstract interpretation the best correct approximation of a semantic function f on an abstract domain A that is specified through abstraction/concretization maps α/γ is given by $f^A \triangleq \alpha \circ f \circ \gamma$. Consider a simple operation of increment x++ on an integer variable x. In this case, the best correct approximations on A_1 and A_2 are as follows:

$$\text{++}^{A_1} = \{0 \mapsto \mathbb{Z}_{\geq 0},\ \mathbb{Z}_{\leq 0} \mapsto \mathbb{Z},\ \mathbb{Z}_{\geq 0} \mapsto \mathbb{Z}_{\geq 0},\ \mathbb{Z} \mapsto \mathbb{Z}\},$$
$$\text{++}^{A_2} = \{\mathbb{Z}_{\geq 0} \mapsto \mathbb{Z}_{\geq 0},\ \mathbb{Z} \mapsto \mathbb{Z}\}.$$

We observe that the best correct approximations of ++ in A_1 and A_2 encode the same function, meaning that the approximations of ++ in A_1 and A_2 are equivalent, the latter

being clearly simpler. In other terms, the abstract domain A_1 contains some "irrelevant" elements for approximating the increment operation, that is, 0 and $\mathbb{Z}_{\leq 0}$. We formalize this simplification of an abstract domain relatively to a semantic function in the most general abstract interpretation setting. This allows us to provide, for generic continuous semantic functions, a systematic and constructive method, that we call *correctness kernel*, for simplifying a given abstraction A relatively to a given semantic function f towards the unique minimal abstract domain that induces an equivalent approximate behavior of f as in A. EGAS is then designed by iteratively applying the correctness kernel to abstractions. We show how EGAS can be embedded within the CEGAR methodology by providing a novel refinement heuristics in a CEGAR iteration step which turns out to be more accurate than the basic refinement heuristics [5]. We also describe how EGAS may be applied in predicate abstraction-based model checking [11,18] for reducing the search space without applying Ball et al.'s [2] Cartesian abstractions, which typically yield additional loss of precision.

2 Correctness Kernels

As usual in standard abstract interpretation [8,9], abstract domains (or abstractions) are specified by Galois connections/insertions (GCs/GIs for short). A GC/GI of the abstract domain A into the concrete domain C through abstraction and concretization maps $\alpha : C \to A$ and $\gamma : A \to C$ is denoted by (α, C, A, γ). It is known that $\mu_A \triangleq \gamma \circ \alpha : C \to C$ is an upper closure operator (uco) on C and that abstract domains can be equivalently defined as uco's. GIs of a common concrete domain C are preordered w.r.t. precision as usual: $\mathcal{G}_1 = (\alpha_1, C, A_1, \gamma_1) \sqsubseteq \mathcal{G}_2 = (\alpha_2, C, A_2, \gamma_2)$ — i.e. A_1/A_2 is a refinement/simplification of A_2/A_1 — iff $\gamma_1 \circ \alpha_1 \sqsubseteq \gamma_2 \circ \alpha_2$. Moreover, $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent when $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}_1$. We denote by $\mathrm{Abs}(C)$ the family of abstract domains of C up to the above equivalence. It is well known that $\langle \mathrm{Abs}(C), \sqsubseteq \rangle$ is a complete lattice, so that one can consider the most concrete simplification (i.e., lub $\sqcup$) and the most abstract refinement (i.e., glb $\sqcap$) of any family of abstract domains. Let us also recall that the lattice of abstract domains $\langle \mathrm{Abs}(C), \sqsubseteq \rangle$ is isomorphic to the lattice of uco's on C $\langle \mathrm{uco}(C), \sqsubseteq \rangle$.

Let $A \in \mathrm{Abs}(C)$, $f : C \to C$ be some concrete semantic function — for simplicity, we consider 1-ary functions — and $f^\sharp : A \to A$ be a corresponding abstract function. $\langle A, f^\sharp \rangle$ is a sound abstract interpretation when $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$. The abstract function $f^A \triangleq \alpha \circ f \circ \gamma : A \to A$ is called the best correct approximation (b.c.a.) of f on A because $\langle A, f^\sharp \rangle$ is sound iff $f^A \sqsubseteq f^\sharp$.

2.1 The Problem

Given a semantic function $f : C \to C$ and an abstract domain $A \in \mathrm{Abs}(C)$, does there exist the *most abstract domain* that induces the same best correct approximation of f as A does?

Let us formalize the above question. Consider two abstractions $A, B \in \mathrm{Abs}(C)$. We say that A and B induce the same best correct approximation of f when f^A and f^B are the same function up to isomorphic representations of abstract values, i.e., if μ_A and

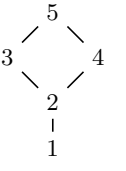
μ_B are the corresponding uco's then $\mu_A \circ f \circ \mu_A = \mu_B \circ f \circ \mu_B$. In order to keep the notation easy, this is denoted simply by $f^A = f^B$. Also, if $F \subseteq C \longrightarrow C$ is a set of concrete functions then $F^A = F^B$ means that for any $f \in F$, $f^A = f^B$. Hence, given $A \in \mathrm{Abs}(C)$ and by defining

$$A_s \triangleq \sqcup\{B \in \mathrm{Abs}(C) \mid F^B = F^A\}$$

the question is whether $F^{A_s} = F^A$ holds or not.

It is worth remarking that the dual question on the existence of the *most concrete domain* that induces the same best correct approximation of f as A has a negative answer, as shown by the following simple example.

Example 2.1. Consider the lattice C depicted on the right, the monotonic function $f : C \to C$ defined as $f \triangleq \{1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 5, 4 \mapsto 5, 5 \mapsto 5\}$ and the domain $\mu \in \mathrm{uco}(C)$ defined as $\mu \triangleq \{1, 5\}$. We have that $\mu \circ f \circ \mu = \{1 \mapsto 1, 2 \mapsto 5, 3 \mapsto 5, 4 \mapsto 5, 5 \mapsto 5\}$. Consider the domains $\rho_1 \triangleq \{1, 3, 5\}$ and $\rho_2 \triangleq \{1, 4, 5\}$ and observe that $\rho_i \circ f \circ \rho_i = \mu \circ f \circ \mu$. However, $\rho_1 \sqcap \rho_2 = \lambda x.x$, so that $(\rho_1 \sqcap \rho_2) \circ f \circ (\rho_1 \sqcap \rho_2) = f \neq \mu \circ f \circ \mu$. Thus, we have that the most concrete domain that induces the same best correct approximation of f as μ does not exist. □



2.2 The Solution

Our key technical result is the following *constructive* characterization of the property of "having the same b.c.a." for two comparable abstract domains. In the following, given a poset A and $S \subseteq A$, $\max(S) \triangleq \{x \in S \mid \forall y \in S.\ x \leq_A y \Rightarrow x = y\}$ denotes the set of maximal elements of S in A.

Lemma 2.2. *Let $f : C \to C$, $A, B \in \mathrm{Abs}(C)$ such that $B \subseteq A$ and $f \circ \mu_A$ is continuous (i.e., preserves lub's of chains). Then,*

$$f^B = f^A \Leftrightarrow \mathrm{img}(f^A) \cup \bigcup_{y \in A} \max(\{x \in A \mid f^A(x) \leq_A y\}) \subseteq B.$$

It is important to remark that the proof of the above result basically consists in reducing the equality $f^A = f^B$ between b.c.a.'s to a standard property of completeness of the abstract domains A and B for the function f and then in exploiting the constructive characterization of completeness of abstract domains by Giacobazzi et al. [17, Section 4]. In this sense, the proof itself is particularly interesting because it provides an unexpected reduction of best correct approximations as a completeness problem.

Definition 2.3. Given $F \subseteq C \longrightarrow C$ define: $\mathcal{K}_F : \mathrm{Abs}(C) \to \mathrm{Abs}(C)$ as

$$\mathcal{K}_F(A) \triangleq \sqcup\{B \in \mathrm{Abs}(C) \mid F^B = F^A\}.$$

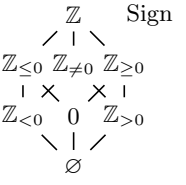
If $F^{\mathcal{K}_F(A)} = F^A$ then $\mathcal{K}_F(A)$ is called the *correctness kernel* of A for F. □

As a consequence of Lemma 2.2 we obtain the following constructive result of existence for correctness kernels. If $X \subseteq A$ then $\mathrm{Cl}_\wedge(X) \triangleq \{\wedge S \in A \mid S \subseteq X\}$ denotes the glb-closure of X in A (note that $\wedge\varnothing = \top_A \in \mathrm{Cl}_\wedge(X)$), while $\mathrm{Cl}_\vee(X)$ denotes the dual lub-closure.

Theorem 2.4. *Let $A \in \mathrm{Abs}(C)$ and $F \subseteq C \to C$ such that, for any $f \in F$, $f \circ \mu_A$ is continuous. Then, the correctness kernel of A for F exists and it is*

$$\mathcal{K}_F(A) = \mathrm{Cl}_\wedge\Big(\bigcup_{f\in F} \mathrm{img}(f^A) \cup \bigcup_{y\in\mathrm{img}(f^A)} \max(\{x \in A \mid f^A(x) = y\})\Big).$$

Example 2.5. Consider sets of integers $\wp(\mathbb{Z})_\subseteq$ as concrete domain and the square operation $sq : \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ as concrete function, i.e., $sq(X) \triangleq \{x^2 \mid x \in X\}$, which is obviously additive and therefore continuous. Consider the abstract domain $\mathrm{Sign} \in \mathrm{Abs}(\wp(\mathbb{Z})_\subseteq)$, depicted in the figure, that represents the sign of an integer variable. Sign induces the following best correct approximation of sq:



$$sq^{\mathrm{Sign}} = \{\varnothing \mapsto \varnothing, \mathbb{Z}_{<0} \mapsto \mathbb{Z}_{>0}, 0 \mapsto 0, \mathbb{Z}_{>0} \mapsto \mathbb{Z}_{>0}, \mathbb{Z}_{\leq 0} \mapsto \mathbb{Z}_{\geq 0},$$
$$\mathbb{Z}_{\neq 0} \mapsto \mathbb{Z}_{>0}, \mathbb{Z}_{\geq 0} \mapsto \mathbb{Z}_{\geq 0}, \mathbb{Z} \mapsto \mathbb{Z}_{\geq 0}\}.$$

Let us characterize the correctness kernel $\mathcal{K}_{sq}(\mathrm{Sign})$ by Theorem 2.4. We have that $\mathrm{img}(sq^{\mathrm{Sign}}) = \{\varnothing, \mathbb{Z}_{>0}, 0, \mathbb{Z}_{\geq 0}\}$. Moreover,

$$\max(\{x \in \mathrm{Sign} \mid sq^{\mathrm{Sign}}(x) = \varnothing\}) = \{\varnothing\}$$
$$\max(\{x \in \mathrm{Sign} \mid sq^{\mathrm{Sign}}(x) = \mathbb{Z}_{>0}\}) = \{\mathbb{Z}_{\neq 0}\}$$
$$\max(\{x \in \mathrm{Sign} \mid sq^{\mathrm{Sign}}(x) = 0\}) = \{0\}$$
$$\max(\{x \in \mathrm{Sign} \mid sq^{\mathrm{Sign}}(x) = \mathbb{Z}_{\geq 0}\}) = \{\mathbb{Z}\}$$

Hence, $\cup_{y\in\mathrm{img}(sq^{\mathrm{Sign}})} \max(\{x \in \mathrm{Sign} \mid sq^{\mathrm{Sign}}(x) = y\}) = \{\varnothing, \mathbb{Z}_{\neq 0}, 0, \mathbb{Z}\}$ so that, by Theorem 2.4:

$$\mathcal{K}_{sq}(\mathrm{Sign}) = \mathrm{Cl}_\cap(\{\varnothing, \mathbb{Z}_{>0}, 0, \mathbb{Z}_{\geq 0}, \mathbb{Z}_{\neq 0}, \mathbb{Z}\}) = \mathrm{Sign} \setminus \{\mathbb{Z}_{<0}, \mathbb{Z}_{\leq 0}\}.$$

Thus, it turns out that we can safely remove the abstract values $\mathbb{Z}_{<0}$ and $\mathbb{Z}_{\leq 0}$ from Sign and still preserve the same b.c.a. as Sign does. Besides, we cannot remove further abstract elements otherwise we do not retain the same b.c.a. as Sign. For example, this means that Sign-based analyses of programs like

$$x := k; \textbf{while } \text{condition } \textbf{do } x := x * x;$$

can be carried out by using the simpler domain $\mathrm{Sign} \setminus \{\mathbb{Z}_{<0}, \mathbb{Z}_{\leq 0}\}$, providing the same input/output abstract behavior. □

It is also worth remarking that in Theorem 2.4 the hypothesis of continuity is crucial for the existence of correctness kernels as shown by the following example.

Example 2.6. Let us consider as concrete domain C the $\omega + 2$ ordinal, i.e., $C \triangleq \{x \in Ord \mid x < \omega\} \cup \{\omega, \omega + 1\}$, and let $f : C \longrightarrow C$ be defined as follows: for any $x < \omega$, $f(x) \triangleq \omega$, and $f(\omega) = f(\omega + 1) \triangleq \omega + 1$. Let $\mu \in \mathrm{uco}(C)$ be the identity $\lambda x.x$, so that $\mu \circ f \circ \mu = f$. For any $k \geq 0$, consider $\rho_k \in \mathrm{uco}(C)$ defined as $\rho_k \triangleq C \smallsetminus [0, k[$ and observe that for any k, we have that $\rho_k \circ f \circ \rho_k = f = \mu \circ f \circ \mu$. However, it turns out that $\sqcup_{k \geq 0} \rho_k = \{\omega, \omega + 1\}$ and $(\sqcup_{k \geq 0} \rho_k) \circ f \circ (\sqcup_{k \geq 0} \rho_k) = \lambda x.\omega + 1 \neq \mu \circ f \circ \mu$. Hence, the correctness kernel of μ for f does not exist. Observe that $\mu \circ f = f$ is not continuous and therefore this example is consistent with Theorem 2.4. □

3 Correctness Kernels in Abstract Model Checking

Following [19], partitions can be viewed as particular abstract domains of the concrete domain $\wp(\Sigma)$. Let $\mathrm{Part}(\Sigma)$ denote the set of partitions of Σ. A partition $P \in \mathrm{Part}(\Sigma)$ can be considered an abstract domain by means of the following Galois insertion $(\alpha_P, \wp(\Sigma)_\subseteq, \wp(P)_\subseteq, \gamma_P)$: $\alpha_P(S) \triangleq \{B \in P \mid B \cap S \neq \varnothing\}$ and $\gamma_P(\mathcal{B}) \triangleq \cup_{B \in \mathcal{B}} B$. Hence, $\alpha_P(S)$ encodes the minimal over-approximation of a set S of states through blocks of P.

Consider a Kripke structure $\mathcal{K} = \langle \Sigma, \rightarrow, \ell \rangle$ and a corresponding abstract Kripke structure $\mathcal{A} = \langle P, \rightarrow^\sharp, \ell^\sharp \rangle$ defined over a state partition $P \in \mathrm{Part}(\Sigma)$.[1] Fixpoint-based verification of a temporal specification on the abstract model $\mathcal{A}$ involves the computation of least/greatest fixpoints of operators defined using Boolean connectives (union, intersection, complementation) on abstract states and abstract successor/predecessor functions $\mathrm{post}^\sharp/\mathrm{pre}^\sharp$ on the abstract transition system $\langle P, \rightarrow^\sharp \rangle$. The key point here is that sucessor/predecessor functions are defined as best correct approximations on the abstract domain P of the corresponding concrete successor/predecessor functions. In standard abstract model checking [1,6,7], the abstract transition relation is defined as the existential/existential relation $\rightarrow^{\exists\exists}$ between blocks of P:

$$B \rightarrow^{\exists\exists} C \text{ iff } \exists x \in B. \exists y \in C.\ x \rightarrow y$$
$$\mathrm{post}^{\exists\exists}(B) \triangleq \{C \in P \mid B \rightarrow^{\exists\exists} C\};\ \mathrm{pre}^{\exists\exists}(C) \triangleq \{B \in P \mid B \rightarrow^{\exists\exists} C\}.$$

It turns out [19] that $\mathrm{pre}^{\exists\exists}$ and $\mathrm{post}^{\exists\exists}$ are the best correct approximations of, resp., pre and post on the abstract domain $(\alpha_P, \wp(\Sigma)_\subseteq, \wp(P)_\subseteq, \gamma_P)$. In fact, for a block $C \in P$,

$$\alpha_P(\mathrm{pre}(\gamma_P(C))) = \{B \in P \mid B \cap \mathrm{pre}(C) \neq \varnothing\} = \mathrm{pre}^{\exists\exists}(C)$$

and an analogous equation holds for post. We thus have that $\mathrm{pre}^{\exists\exists} = \alpha_P \circ \mathrm{pre} \circ \gamma_P$ and $\mathrm{post}^{\exists\exists} = \alpha_P \circ \mathrm{post} \circ \gamma_P$.

This abstract interpretation-based framework allows us to apply correctness kernels in the context of abstract model checking. The abstract transition system $\mathcal{A} = \langle P, \rightarrow^{\exists\exists} \rangle$ is viewed as the abstract interpretation defined by the abstract domain $(\alpha_P, \wp(\Sigma)_\subseteq, \wp(P)_\subseteq, \gamma_P)$ and the corresponding abstract functions $\mathrm{pre}^{\exists\exists} = \alpha_P \circ \mathrm{pre} \circ \gamma_P$ and $\mathrm{post}^{\exists\exists} = \alpha_P \circ \mathrm{post} \circ \gamma_P$. Then, the correctness kernel of $\wp(P)$ for the concrete predecessor/successor $\{\mathrm{pre}, \mathrm{post}\}$, that we denote simply by $\mathcal{K}_\rightarrow(P)$ (this clearly exists

[1] Equivalently, the abstract Kripke structure $\mathcal{A}$ can be defined over an abstract state space A determined by a surjective abstraction function $h : \Sigma \longrightarrow A$.

by Theorem 2.4 since pre and post are additive), provides a simplification of the abstract domain $\wp(P)$ that preserves the best correct approximations of predecessor and successor. This simplification of the abstract state space works as follows:

Corollary 3.1. *$\mathcal{K}_{\rightarrow}(P)$ merges two blocks $B_1, B_2 \in P$ if and only if for any $A \in P$, $A \rightarrow^{\exists\exists} B_1 \Leftrightarrow A \rightarrow^{\exists\exists} B_2$ and $B_1 \rightarrow^{\exists\exists} A \Leftrightarrow B_2 \rightarrow^{\exists\exists} A$.*

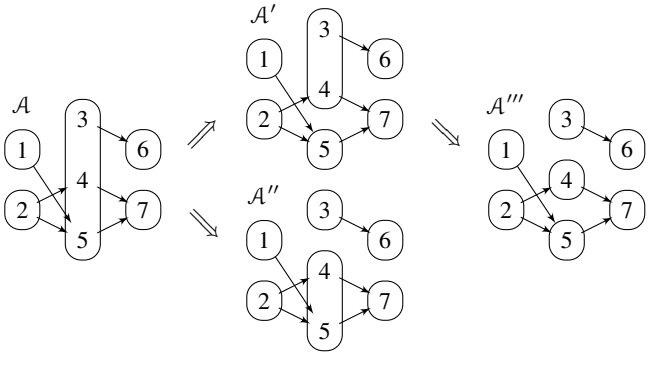
4 EGAS and CEGAR

Let us discuss how correctness kernels give rise to an Example-Guided Abstraction Simplification (EGAS) paradigm in abstract transition systems.

Let us first recall some basic notions of CEGAR [4,5]. Consider an abstract transition system $\mathcal{A} = \langle P, \rightarrow^{\exists\exists} \rangle$ defined over a state partition $P \in \mathrm{Part}(\Sigma)$ and a finite abstract path $\pi = \langle B_1, ..., B_n \rangle$ in $\mathcal{A}$. Typically, this is a path counterexample to the validity of a temporal formula that has been given as output by a model checker (for simplicity we do not consider here loop path counterexamples). The set of concrete paths that are abstracted to π are defined as follows: $\mathrm{paths}(\pi) \triangleq \{\langle s_1, ..., s_n \rangle \in \Sigma^n \mid \forall i \in [1, n].s_i \in B_i \ \& \ \forall i \in [1, n).s_i \rightarrow s_{i+1}\}$. The abstract path π is *spurious* when $\mathrm{paths}(\pi) = \varnothing$. The sequence of sets of states $\mathrm{sp}(\pi) = \langle S_1, ..., S_n \rangle$ is inductively defined as follows: $S_1 \triangleq B_1$; $S_{i+1} \triangleq \mathrm{post}(S_i) \cap B_{i+1}$. As shown in [5], it turns out that π is spurious iff there exists a least $k \in [1, n-1]$ such that $S_{k+1} = \varnothing$. In such a case, the partition P is refined by splitting the block B_k. The three following sets partition the block B_k:

dead-end states: $B_k^{\mathrm{dead}} \triangleq S_k \neq \varnothing$
bad states: $B_k^{\mathrm{bad}} \triangleq B_k \cap \mathrm{pre}(B_{k+1}) \neq \varnothing$
irrelevant states: $B_k^{\mathrm{irr}} \triangleq B_k \setminus (B_k^{\mathrm{dead}} \cup B_k^{\mathrm{bad}})$

The split of the block B_k must separate dead-end states from bad states, while irrelevant states may be joined indifferently with dead-end or bad states. However, the problem of finding the coarsest refinement of P that separates dead-end and bad states is NP-hard [5] and thus some refinement heuristics are used. According to the basic heuristics in [5, Section 4], B_k is simply split into B_k^{dead} and $B_k^{\mathrm{bad}} \cup B_k^{\mathrm{irr}}$. Let us see a simple example.



Consider the abstract path $\pi = \langle [1], [345], [6] \rangle$ in $\mathcal{A}$. This is a spurious path and the block $[345]$ is therefore partitioned as follows: $[5]$ dead-end states, $[3]$ bad states and $[4]$ irrelevant states. The refinement heuristics of CEGAR tells us that irrelevant states are joined with bad states so that $\mathcal{A}$ is refined to $\mathcal{A}'$. In turn, consider the spurious path $\pi' = \langle [2], [34], [6] \rangle$ in $\mathcal{A}'$, so that CEGAR refines $\mathcal{A}'$ to $\mathcal{A}'''$ by splitting the block $[34]$. In the first abstraction refinement, let us observe that if irrelevant states would have been joined together with dead-end states rather than with bad states we would have obtained the abstract system $\mathcal{A}''$, and $\mathcal{A}''$ does not contain spurious paths so that it surely does not need to be further refined.

EGAS can be integrated within the CEGAR loop thanks to the following remark. If π_1 and π_2 are paths, resp., in $\langle P_1, \rightarrow^{\exists\exists} \rangle$ and $\langle P_2, \rightarrow^{\exists\exists} \rangle$, where $P_1, P_2 \in \mathrm{Part}(\Sigma)$ and $P_1 \preceq P_2$, then π_1 is abstracted to π_2, denoted by $\pi_1 \sqsubseteq \pi_2$, when $\mathrm{length}(\pi_1) = \mathrm{length}(\pi_2)$ and for any $j \in [1, \mathrm{length}(\pi_1)]$, $\pi_1(j) \subseteq \pi_2(j)$.

Corollary 4.1. *Consider an abstract transition system $\mathcal{A} = \langle P, \rightarrow^{\exists\exists} \rangle$ over a partition $P \in \mathrm{Part}(\Sigma)$ and its simplification $\mathcal{A}_s = \langle \mathcal{K}_{\rightarrow}(P), \rightarrow^{\exists\exists} \rangle$ induced by the correctness kernel $\mathcal{K}_{\rightarrow}(P)$. If π is an abstract path in $\mathcal{A}_s$ such that $\mathrm{paths}(\pi) = \varnothing$ then there exists an abstract path π' in $\mathcal{A}$ such that $\pi' \sqsubseteq \pi$ and $\mathrm{paths}(\pi') = \varnothing$.*

Thus, the abstraction simplification induced by the correctness kernel does not add spurious paths.

The above observations suggest us a new refinement strategy within the CEGAR loop. Let $\pi = \langle B_1, ..., B_n \rangle$ be a spurious path in $\mathcal{A}$ and $\mathrm{sp}(\pi) = \langle S_1, ..., S_n \rangle$ such that $S_{k+1} = \varnothing$ for some minimum $k \in [1, n-1]$, so that the block B_k needs to be split. The set of irrelevant states B_k^{irr} is partitioned as follows. We first define the subset of *bad-irrelevant* states $B_k^{\text{bad-irr}}$. Let $\mathrm{pre}^{\exists\exists}(B_k^{\text{bad}}) = \{A_1, ..., A_j\}$ and $\mathrm{post}^{\exists\exists}(B_k^{\text{bad}}) = \{C_1, ..., C_l\}$. Then,

$$B_k^{\text{bad-irr}} \triangleq \big(\mathrm{post}(A_1 \cup ... \cup A_j) \cap \mathrm{pre}(C_1 \cup ... \cup C_l)\big) \cap B_k^{\text{irr}}.$$

The underlying idea is simple: $B_k^{\text{bad-irr}}$ contains the irrelevant states that: (1) can be reached from a block that reaches some bad state and (2) reach a block that is also reached by some bad state. By Corollary 4.1, it is therefore clear that by merging $B_k^{\text{bad-irr}}$ and B_k^{bad} no spurious path is added w.r.t. the abstract system where they are kept separate. The subset of *dead-irrelevant* states $B_k^{\text{dead-irr}}$ is analogosly defined: If $\mathrm{pre}^{\exists\exists}(B_k^{\text{dead}}) = \{A_1, ..., A_j\}$ and $\mathrm{post}^{\exists\exists}(B_k^{\text{dead}}) = \{C_1, ..., C_l\}$ then

$$B_k^{\text{dead-irr}} \triangleq \big(\mathrm{post}(A_1 \cup ... \cup A_j) \cap \mathrm{pre}(C_1 \cup ... \cup C_l)\big) \cap B_k^{\text{irr}}.$$

It may happen that: (A) an irrelevant state is both bad- and dead-irrelevant; (B) an irrelevant state is neither bad- nor dead-irrelevant. From the viewpoint of EGAS, the states of case (A) can be equivalently merged with bad or dead states since in both cases no spurious path is added. On the other hand, the states of case (B) are called *fully-irrelevant* because EGAS does not provide a merging strategy with bad or dead states. For these states, one could use, for example, the basic refinement heuristics of CEGAR that merge them with bad states.

In the above example, for the spurious path $\langle [1], [345], [6] \rangle$ in $\mathcal{A}$, the block $B = [345]$ needs to be refined: $B^{\text{bad}} = [3]$, $B^{\text{dead}} = [5]$ and $B^{\text{irr}} = [4]$. Here, 4 is a dead-irrelevant state and so it is merged in $\mathcal{A}''$ with the dead-end state 5.

5 Correctness Kernels in Predicate Abstraction

Let us show how correctness kernels can be also used in the context of predicate abstraction-based model checking [11,18]. Following Ball et al. [2], predicate abstraction can be formalized by abstract interpretation as follows. Let us consider a program P with k integer variables x_1,...,x_k. The concrete domain of computation of P is $\langle \wp(\mathrm{States}), \subseteq \rangle$ where $\mathrm{States} \triangleq \{x_1, ..., x_k\} \to \mathbb{Z}$. Values in States are denoted by tuples $\langle z_1, ..., z_k \rangle \in \mathbb{Z}^k$. The program P generates a transition system $\langle \mathrm{States}, \to \rangle$ so that the concrete semantics of P is defined by the corresponding successor function $\mathrm{post} : \wp(\mathrm{States}) \to \wp(\mathrm{States})$.

A finite set $\mathcal{P} = \{p_1, ..., p_n\}$ of state predicates is considered, where each predicate p_i denotes the subset of states that satisfy p_i, i.e. $\{s \in \mathrm{States} \mid s \models p_i\}$. These predicates give rise to the so-called Boolean abstraction $B \triangleq \langle \wp(\{0,1\}^n), \subseteq \rangle$ which is related to $\wp(\mathrm{States})$ through the following abstraction/concretization maps (here, $s \models p_i$ is understood in $\{0,1\}$) that give rise to a disjunctive (i.e., γ preserves lub's) Galois connection:

$$\alpha_B(S) \triangleq \{\langle s \models p_1, ..., s \models p_n \rangle \in \{0,1\}^n \mid s \in S\},$$
$$\gamma_B(V) \triangleq \{s \in \mathrm{States} \mid \langle s \models p_1, ..., s \models p_n \rangle \in V\}.$$

Verification of reachability properties based on predicate abstraction consists in computing the least fixpoint of the best correct approximation of post on the Boolean abstraction B, namely, $\mathrm{post}^B \triangleq \alpha_B \circ \mathrm{post} \circ \gamma_B$. As argued in [2], the Boolean abstraction B may be too costly for the purpose of reachability verification, so that one usually abstracts B through the so-called Cartesian abstraction. This latter abstraction formalizes precisely the abstract post operator computed by the verification algorithm of the c2bp tool in SLAM [3]. However, the Cartesian abstraction of B may cause a loss of precision, so that this abstraction is successively refined by reduced disjunctive completion and the so-called focus operation, and this formalizes the bebop tool in SLAM [2].

Let us consider the example program in Figure 1, taken from [2], where the goal is that of verifying that the assert at line $(*)$ is never reached, regardless of the context in which $foo()$ is called. Ball et al. [2] consider the following set of predicates

```
int x, y, z, w;
void foo() {
  do {
    z := 0; x := y;
    if (w) { x++; z := 1; }
  } while (!(x = y))
  if (z)
    assert(0);  // (*)
}
```

Fig. 1. An example program

$\mathcal{P} \triangleq \{p_1 \equiv (z = 0), p_2 \equiv (x = y)\}$ so that the Boolean abstraction is $B = \wp(\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\})_{\subseteq}$. Clearly, the analysis based on B allows to conclude that line $(*)$ is not reachable. This comes as a consequence of the fact that the least fixpoint of the best correct approximation post^B for the do-while-loop provides as result $\{\langle 0,0\rangle, \langle 1,1\rangle\} \in B$ because:

$$\varnothing \xrightarrow{z:=0;\ x:=y} \{\langle 1,1\rangle\} \xrightarrow{\textbf{if}(w)\{x{+}{+};\ z:=1;\}} \{\langle 1,1\rangle, \langle 0,0\rangle\}$$

so that at the exit of the do-while-loop one can conclude that

$$\{\langle 1,1\rangle, \langle 0,0\rangle\} \cap p_2 = \{\langle 1,1\rangle, \langle 0,0\rangle\} \cap \{\langle 0,1\rangle, \langle 1,1\rangle\} = \{\langle 1,1\rangle\}$$

holds, hence p_1 is satisfied, so that $z = 0$ and therefore line $(*)$ cannot be reached.

Let us characterize the correctness kernel of the Boolean abstraction B. Let $S_1 \triangleq z := 0;\ x := y$ and $S_2 \triangleq x{+}{+};\ z := 1$. The best correct approximations of post_{S_1} and post_{S_2} on the abstract domain B turn out to be as follows:

$$\alpha_B \circ \mathrm{post}_{S_1} \circ \gamma_B = \Big\{\langle 0,0\rangle \mapsto \{\langle 1,1\rangle\}, \langle 0,1\rangle \mapsto \{\langle 1,1\rangle\}, \langle 1,0\rangle \mapsto \{\langle 1,1\rangle\}, \langle 1,1\rangle \mapsto \{\langle 1,1\rangle\}\Big\}$$

$$\alpha_B \circ \mathrm{post}_{S_2} \circ \gamma_B = \Big\{\langle 0,0\rangle \mapsto \{\langle 0,0\rangle, \langle 0,1\rangle\}, \langle 0,1\rangle \mapsto \{\langle 0,0\rangle\}, \langle 1,0\rangle \mapsto \{\langle 0,0\rangle, \langle 0,1\rangle\}, \langle 1,1\rangle \mapsto \{\langle 0,0\rangle\}\Big\}$$

Thus, we have that $\mathrm{img}(\alpha_B \circ \mathrm{post}_{S_1} \circ \gamma_B) = \{\{\langle 1,1\rangle\}\}$ and $\mathrm{img}(\alpha_B \circ \mathrm{post}_{S_2} \circ \gamma_B) = \{\{\langle 0,0\rangle, \langle 0,1\rangle\}, \{\langle 0,0\rangle\}\}$ so that

$$\max\big(\{V \in B \mid \alpha_B(\mathrm{post}_{S_1}(\gamma_B(V))) = \{\langle 1,1\rangle\}\}\big) = \{\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}\}$$
$$\max\big(\{V \in B \mid \alpha_B(\mathrm{post}_{S_2}(\gamma_B(V))) = \{\langle 0,0\rangle, \langle 0,1\rangle\}\}\big) = \{\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}\}$$
$$\max\big(\{V \in B \mid \alpha_B(\mathrm{post}_{S_2}(\gamma_B(V))) = \{\langle 0,0\rangle\}\}\big) = \{\{\langle 0,1\rangle, \langle 1,1\rangle\}\}$$

Hence, by Theorem 2.4, the kernel $\mathcal{K}_F(B)$ of B for $F \triangleq \{\mathrm{post}_{S_1}, \mathrm{post}_{S_2}\}$ is:

$$\mathrm{Cl}_\cap\Big(\mathrm{Cl}_\cup\big(\{\{\langle 0,0\rangle\}, \{\langle 1,1\rangle\}, \{\langle 0,0\rangle, \langle 0,1\rangle\}, \{\langle 0,1\rangle, \langle 1,1\rangle\}, \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}\}\big)\Big) = \mathrm{Cl}_\cup\Big(\big\{\{\langle 0,0\rangle\}, \{\langle 0,1\rangle\}, \{\langle 1,1\rangle\}\big\}\Big)$$

This means that $\mathcal{K}_F(B)$ can be represented as

$$\langle \wp(\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,1\rangle\}) \cup \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}, \subseteq\rangle$$

and therefore $\mathcal{K}_F(B)$ is a proper abstraction of B that, for example, is not able to express the property $p_1 \wedge \neg p_2 \equiv (z = 0) \wedge (x \neq y)$.

It is interesting to compare this correctness kernel $\mathcal{K}_F(B)$ with Ball et al.'s [2] Cartesian abstraction of B. The Cartesian abstraction is defined as $C \triangleq \langle \{0,1,*\}^n \cup \{\bot_C\}, \leq\rangle$, where $\leq$ is pointwise ordering between tuples of values in $\{0,1,*\}$ ordered by $0 < *$ and $1 < *$ ($\bot$ is the bottom element that represents the empty set of states). The concretization function $\gamma_C : C \to \wp(\mathrm{States})$ is as follows:

$$\gamma_C(\langle v_1, ..., v_n\rangle) \triangleq \{s \in \text{States} \mid \langle s \models p_1, ..., s \models p_n\rangle \leq \langle v_1, ..., v_n\rangle\}.$$

These two abstractions are not comparable: for instance, $\langle 1, 0\rangle \in C$ represents $p_1 \wedge \neg p_2$ which is instead not represented by $\mathcal{K}_F(B)$, while $\{\langle 0, 0\rangle, \langle 1, 1\rangle\} \in \mathcal{K}_F(B)$ represents $(\neg p_1 \wedge \neg p_2) \vee (p_1 \wedge p_2)$ which is not represented in C. However, while the correctness kernel guarantees no loss of information in analyzing the program P, the analysis of P with the Cartesian abstraction C is inconclusive because:

$$\bot_C \xrightarrow{z:=0;\ x:=y} \langle 1, 1\rangle \xrightarrow{\textbf{if}(w)\{x{+}{+};\ z:=1;\}} \langle 0, 0\rangle \vee_C \langle 1, 1\rangle = \langle *, *\rangle$$

where $\gamma_C(\langle *, *\rangle) = \text{States}$, so that at the exit of the do-while-loop one cannot infer that line $(*)$ cannot be reached.

6 Related and Future Work

Few examples of abstraction simplifications are known. A general notion of domain simplification and compression in abstract interpretation has been introduced in [12,15] as a dual of abstraction refinement. This duality has been further exploited in [13] to include semantics transformations in a general theory for transforming abstractions and semantics based on abstract interpretation. Our domain transformation does not fit directly in this framework. Following [15], given a property $\mathcal{P}$ of abstract domains, the core of an abstract domain A, when it exists, provides the most concrete simplification of A that satisfies the property $\mathcal{P}$, while the compressor of A, when it exists, provides the most abstract simplification of A that induces the same refined abstraction in $\mathcal{P}$ as A does. Examples of compressors include the least disjuctive basis [16], where $\mathcal{P}$ is the abstract domain property of being disjunctive, and examples of cores include the completeness core [17], where $\mathcal{P}$ is the domain property of being complete for some semantic function. The correctness kernel defined in this paper is neither an instance of a domain core nor an instance of domain compression. The first because, given an abstraction A, the correctness kernel of A characterizes the most abstract domain that induces the same best correct approximation of a function f on A, whilst the notion of domain core for the domain property $\mathcal{P}_A$ of inducing the same b.c.a. as A would not be meaningful, as this would trivially yield A itself. The second because there is no (unique) maximal domain refinement of an abstract domain which induces the same property $\mathcal{P}_A$.

The EGAS methodology opens some stimulating directions for future work, such as (1) the formalization of a precise relationship between EGAS and CEGAR and (2) an experimental evaluation of the integration in the CEGAR loop of the EGAS-based refinement strategy of Section 4. It is here useful to recall that some work formalizing CEGAR in abstract interpretation has already been done [10,14]. On the one hand, [14] shows that CEGAR corresponds to iteratively compute a so-called complete shell [17] of the underlying abstract model A with respect to the concrete successor transformer, while [10] formally compares CEGAR with an abstraction refinement strategy based on the computations of abstract fixpoints in an abstract domain. These works can therefore provide a starting point for studying the relationship between EGAS and CEGAR in a common abstract interpretation setting.

Acknowledgements. This work was carried out during a visit of the authors to the Equipe "Abstraction" lead by P. and R. Cousot, at École Normale Supérieure, Paris. This work was partially supported by the PRIN 2007 Project "*AIDA2007: Abstract Interpretation Design and Applications*" and by the University of Padova under the Project "*Analysis, verification and abstract interpretation of models for concurrency*".

References

1. Baier, C., Katoen, J.-P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
2. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and Cartesian abstraction for model checking C programs. Int. J. Softw. Tools Technol. Transfer 5, 49–58 (2003)
3. Ball, T., Rajamani, S.K.: The SLAM Project: Debugging system software via static analysis. In: Proc. 29th ACM POPL, pp. 1–3 (2002)
4. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
6. Clarke, E.M., Grumberg, O., Long, D.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
8. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. 4th ACM POPL, pp. 238–252 (1977)
9. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. 6th ACM POPL, pp. 269–282 (1979)
10. Cousot, P., Ganty, P., Raskin, J.-F.: Fixpoint-guided abstraction refinements. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 333–348. Springer, Heidelberg (2007)
11. Das, S., Dill, D.L., Park, S.: Experience with predicate abstraction. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 160–171. Springer, Heidelberg (1999)
12. Filé, G., Giacobazzi, R., Ranzato, F.: A unifying view of abstract domain design. ACM Comp. Surveys 28(2), 333–336 (1996)
13. Giacobazzi, R., Mastroeni, I.: Transforming abstract interpretations by abstract interpretation (Invited Lecture). In: Alpuente, M., Vidal, G. (eds.) SAS 2008. LNCS, vol. 5079, pp. 1–17. Springer, Heidelberg (2008)
14. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples, and refinements in abstract model checking. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, pp. 356–373. Springer, Heidelberg (2001)
15. Giacobazzi, R., Ranzato, F.: Refining and compressing abstract domains. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 771–781. Springer, Heidelberg (1997)
16. Giacobazzi, R., Ranzato, F.: Optimal domains for disjunctive abstract interpretation. Sci. Comp. Program. 32, 177–210 (1998)
17. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. J. ACM 47(2), 361–416 (2000)
18. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
19. Ranzato, F., Tapparo, F.: Generalized strong preservation by abstract interpretation. J. Logic and Computation 17(1), 157–197 (2007)