Complete Abstractions Everywhere

Francesco Ranzato

Dipartimento di Matematica, University of Padova, Italy

Abstract. While soundness captures an essential requirement of the intrinsic approximation of any static analysis, completeness encodes approximations that are as precise as possible. Although a static analysis of some undecidable program property cannot be complete relatively to its reference semantics, it may well happen that it is complete relatively to an approximated and decidable reference semantics. In this paper, we will argue on the ubiquity of completeness properties in static analysis and we will discuss the beneficial role that completeness can play as a tool for designing and fine-tuning static analyses by reasoning on the completeness properties of their underlying abstract domains.

1 Introduction

Static analysis relies on sound (a.k.a. correct) and computable semantic approximations. A system $\mathcal{S}$ (or some observable behavior of it) is modeled by some semantics $\mathrm{Sem}[\![\mathcal{S}]\!]$ and a static analysis of $\mathcal{S}$ is designed as an approximate semantics $\mathrm{Sem}^\sharp[\![\mathcal{S}]\!]$ which must be sound w.r.t. $\mathrm{Sem}[\![\mathcal{S}]\!]$. This may be called global soundness of static analysis. A system $\mathcal{S}$ is typically designed by some form of composition of a number of constituents c_i and this is reflected on its global semantics $\mathrm{Sem}[\![\mathcal{S}]\!]$ which is commonly defined by some combinations of the semantics $\mathrm{Sem}[\![c_i]\!]$ of its components. Thus, global soundness of a static analysis of $\mathcal{S}$ is ordinarily derived from local soundness of static analyses for its components c_i. This global vs. local picture of the soundness of static analysis is very common, independently of the kind of systems (e.g., programs, reactive systems, hardware systems), of static analysis techniques (dataflow analysis, model checking, abstract interpretation, logical deductive systems, type systems, etc.), of system properties under analysis (safety, liveness, numerical properties, pointer aliasing, type safety, etc.). We might single out a basic and rough proof principle in static analysis: global soundness is derived from local soundness. In particular this applies to static analyses that are designed using some form of abstract interpretation. Let us consider a simplified and recurrent scenario. A constituent of the global semantics is defined as least (or greatest) fixpoint (denoted by lfp) of a monotone function f on some domain of properties D which is endowed with a partial order that encodes relative precision of properties. Here, $\mathrm{lfp}(f)$ play the role of global semantics which is therefore defined through a component $f : D \to D$ that plays the role of local semantics (e.g., f is a transfer function in program analysis). In abstract interpretation, a static analysis is then specified as an abstract fixpoint computation which must be sound for $\mathrm{lfp}(f)$. This is routinely defined through an ordered abstract domain A of properties and an abstract semantic function $f^\sharp : A \to A$ that give rise to a fixpoint-based static analysis $\mathrm{lfp}(f^\sharp)$ (whose decidability and/or practical scalability is usually ensured by chain conditions on A,

widenings/narrowings, interpolations, etc.). Soundness relies on encoding approximation through a concretization map $\gamma : A \to D$ and/or an abstraction map $\alpha : D \to A$: the approximation of some value d through an abstract property a is encoded as $\alpha(d) \leq_A a$ or — equivalently when α/γ form a Galois connection — $d \leq_D \gamma(a)$. Hence, global soundness translates to $\alpha(\mathrm{lfp}(f)) \leq \mathrm{lfp}(f^\sharp)$, local soundness means $\alpha \circ f \leq f^\sharp \circ \alpha$, and the well-known "fixpoint approximation lemma" tells us that local implies global soundness. This fixpoint approximation lemma has been first used in the early abstract interpretation papers [8, Section 9], [6, Chapter 5], [9, Theorem 7.1.0.4], and has been later rediscovered a number of times (e.g., [1, Section 3]).

While (both global and local) soundness captures a basic requirement of any reasonable semantic approximation, completeness encodes approximations that are as precise as possible. Once an abstract domain of properties A has been chosen, A yields a strict upper bound to the precision of static analyses that approximate $\mathrm{Sem}[\![\mathcal{S}]\!]$ over A: this means that $\alpha(\mathrm{Sem}[\![\mathcal{S}]\!])$ represents the best approximation that one can achieve over A. We refer to as *completeness* when $\alpha(\mathrm{Sem}[\![\mathcal{S}]\!]) = \mathrm{Sem}^\sharp[\![\mathcal{S}]\!]$ happens. In fixpoint-based semantics, an abstract interpretation is globally complete when $\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}(f^\sharp)$. Locally for f, completeness means that $\alpha \circ f = f^\sharp \circ \alpha$, meaning that in each local step, $f^\sharp$ behaves in the best possible way as allowed by A. The approximation lemma for soundness becomes the "transfer lemma" for completeness stating the proof principle that local completeness implies global completeness. To the best of our knowledge, this transfer lemma first appears in the early abstract interpretation paper [9, Theorems 7.1.0.4], and has been later re-stated and re-proved multiple times, e.g., [3, Fact 2.3], [4, Lemma 4.3], [1, Section 3] (where it is called μ-fusion rule).

Of course, completeness cannot happen when D is able to represent undecidable properties of $\mathcal{S}$ and the static analysis $\mathrm{Sem}^\sharp[\![\mathcal{S}]\!]$ is computable. Completeness may instead occur and is actually quite common in comparative semantic modeling, i.e., in studying semantic models at different levels of abstraction. For instance, Cousot [7] defines a comprehensive abstract interpretation-based hierarchy of the most common program semantics and highlights where completeness holds. It is important to remark that although a static analysis cannot be complete relatively to its reference semantics, it may well happen that it is complete relatively to an approximated and decidable reference semantics, e.g., as a notable case, a different and more precise static analysis. In this paper, we will argue the important role that completeness can play as a tool for designing and fine-tuning static analyses by reasoning on the completeness properties of their underlying abstract domains. We will provide a number of examples ranging from static analysis to model checking where the view of an abstract domain A under a "completeness light" allows us to understand the deep reason of why and how A actually works.

2 Complete and Exact Abstractions

As mentioned above, static program analysis will be here formalized in a simplified abstract interpretation scenario.

Program semantics. We consider simple while programs with integer variables in *Var*, integer arithmetic expressions in *Aexp* and boolean expressions in *Bexp*. Hence,

$\mathrm{Store} \triangleq \mathit{Var} \to \wp(\mathbb{Z})$ denotes the set of stores, $[\![e]\!] : \mathrm{Store} \to \mathbb{Z}$ the semantics of an arithmetic expression e and $[\![b]\!] : \mathrm{Store} \to \{\mathbf{f}, \mathbf{t}\}$ the semantics of a boolean expression b. Transfer functions $(\!|\cdot|\!) : \wp(\mathrm{Store}) \to \wp(\mathrm{Store})$ model how assignments $x := e$ and boolean tests b affect the store: $(\!|x := e|\!)S \triangleq \{\rho[x \mapsto [\![e]\!]\rho] \in \mathrm{Store} \mid \rho \in S\}$ and $(\!|b|\!)S \triangleq \{\rho \in S \mid [\![b]\!]\rho = \mathbf{t}\}$. The program semantics is usually defined as least fixpoint of a system of recursive equations indexed on program points. We consider here a denotational-style program semantics $[\![P]\!] : \wp(\mathrm{Store}) \to \wp(\mathrm{Store})$ whose definition is standard:

$$[\![\mathit{stm}_1;\ \mathit{stm}_2]\!]S \triangleq [\![\mathit{stm}_2]\!]([\![\mathit{stm}_1]\!]S), \quad [\![\textbf{if } b \textbf{ then } \mathit{stm}]\!]S \triangleq [\![\mathit{stm}]\!](\!|b|\!)S \cup (\!|\neg b|\!)S,$$
$$[\![\textbf{while } b \textbf{ do } \mathit{stm}]\!]S \triangleq (\!|\neg b|\!)\big(\mathrm{lfp}(\lambda T.S \cup [\![\mathit{stm}]\!](\!|b|\!)T)\big).$$

Soundness, Completeness and Exactness. A static program analysis is defined by an abstract semantics of while programs that relies on some store *abstraction*. Given a generic domain of properties D, we denote by $\mathrm{Abs}(D)$ the class of all possible abstractions of D. Thus, given an abstraction $A \in \mathrm{Abs}(\wp(\mathrm{Store}))$ of store properties, a static analysis needs to define *locally sound* abstract transfer functions on A and an abstract program semantics on A which relies on these abstract transfer functions and is required to be *globally sound*. Let us briefly recall how abstractions and local/global soundness are formalized in abstract interpretation.

Abstractions are formalized as domains in abstract interpretation. If $A \in \mathrm{Abs}(D)$ then both D and A are partially ordered by an approximation ordering where $x \leq y$ means that y approximates x (or x is more precise than y). D and A are required to be complete lattices in order to have arbitrary lub's that model concrete and abstract disjunctions. Finally, the abstraction A is related to D at least by a monotone concretization function $\gamma : A \to D$ which provides the concrete meaning of abstract values, so that: (1) $d \leq_D \gamma(a)$ means that a is an abstract approximation of d and (2) $f \circ \gamma \leq_D \gamma \circ f^\sharp$ means that the abstract function $f^\sharp : A \to A$ is a correct (or sound) approximation of the concrete function $f : D \to D$. A concretization function does not ensure the existence of best abstractions in A for concrete properties, i.e., it is not guaranteed that $\wedge_A\{a \in A \mid d \leq_D \gamma(a)\}$ is an abstract approximation of d. Consequently, a concretization function does not guarantee the existence of best correct approximations of concrete functions. Actually, most abstract domains A are related to D also by an abstraction function $\alpha : D \to A$ which provides the best approximation in A of any concrete property. Accordingly, $f^\sharp$ is a correct approximation of f when $\alpha \circ f \leq f^\sharp \circ \alpha$. Clearly, γ and α must agree on the way they encode approximation: this translates to $\alpha(d) \leq_A a \Leftrightarrow d \leq_D \gamma(a)$ and amounts to require that $\langle \alpha, \gamma \rangle$ forms a Galois connection. When f and $f^\sharp$ are monotonic, and therefore have least (and greatest) fixpoints, local soundness for $f^\sharp$ implies global fixpoint soundness $\alpha(\mathrm{lfp}(f)) \leq_A \mathrm{lfp}(f^\sharp)$.

If $f \circ \gamma = \gamma \circ f^\sharp$ then $f^\sharp$ is called an *exact approximation* of f, while $f^\sharp$ is called a *complete approximation* of f when $\alpha \circ f = f^\sharp \circ \alpha$ holds. Here, we have that local exactness/completeness implies global exactness/completeness: If $f^\sharp$ is exact then $\mathrm{gfp}(f) = \gamma(\mathrm{gfp}(f^\sharp))$ and if $f^\sharp$ is complete then $\alpha(\mathrm{lfp}(f)) = \mathrm{lfp}(f^\sharp)$. These can be viewed as two facets of the principle "local implies global completeness" translated in abstract interpretation terms. Exactness and completeness are orthogonal notions. Let us point out that global exactness means that the abstract fixpoint computation $\mathrm{gfp}(f^\sharp)$

yields an exact representation in A of $\mathrm{gfp}(f)$ while global completeness guarantees that the abstract fixpoint computation $\mathrm{lfp}(f^\sharp)$ provides the best representation in A of $\mathrm{lfp}(f)$.

When a Galois connection is assumed, we have that $f^\sharp$ is sound iff $\alpha \circ f \circ \gamma \leq f^\sharp$. Thus, any function f has a best correct approximation in A which is, at least theoretically, defined as $f^{\mathrm{best}_A} \triangleq \alpha \circ f \circ \gamma$. Moreover, it turns out that the possibility of defining an exact or complete abstract function $f^\sharp$ on some abstraction A actually depends only on A itself. In fact, we have that $f^\sharp$ is complete iff $f^\sharp = f^{\mathrm{best}_A}$ and $\alpha \circ f = \alpha \circ f \circ \gamma \circ \alpha$, while $f^\sharp$ is exact iff $f^\sharp = f^{\mathrm{best}_A}$ and $f \circ \gamma = \gamma \circ \alpha \circ f \circ \gamma$. In the following, we thus make reference to a complete and exact abstraction A to mean completeness and exactness of f^{best_A}.

Abstract program semantics. Given a store abstraction $A \in \mathrm{Abs}(\mathrm{Store})$, abstract transfer functions $(\!|\, x := e \,|\!)^A : A \to A$ and $(\!|\, b \,|\!)^A : A \to A$ must be locally sound, i.e. correct: for any set S of stores, $\alpha((\!|\, x := e \,|\!)S) \leq_A (\!|\, x := e \,|\!)^A \alpha(S)$ and $\alpha((\!|\, b \,|\!)S) \leq_A (\!|\, b \,|\!)^A \alpha(S)$. Also observe that the best abstract transfer functions on A are as follows:

$$(\!|\, x := e \,|\!)^{\mathrm{best}_A} a = \alpha(\{\rho[x \mapsto [\![e]\!]\rho] \in \mathrm{Store} \mid \rho \in \gamma(a)\})$$
$$(\!|\, b \,|\!)^{\mathrm{best}_A} a = \alpha(\{\rho \in \gamma(a) \mid [\![b]\!]\rho = \mathbf{t}\})$$

The abstract denotational semantics $[\![P]\!]^A : A \to A$ is derived simply by composing abstract transfer functions and by approximating concrete disjunctions of stores through abstract lub's in A:

$$[\![stm_1;\ stm_2]\!]^A a \triangleq [\![stm_2]\!]^A([\![stm_1]\!]^A a)$$
$$[\![\mathbf{if}\ b\ \mathbf{then}\ stm]\!]^A a \triangleq [\![stm]\!]^A((\!|\, b \,|\!)^A a) \sqcup_A (\!|\, \neg b \,|\!)^A a$$
$$[\![\mathbf{while}\ b\ \mathbf{do}\ stm]\!]^A a \triangleq (\!|\, \neg b \,|\!)^A \big(\mathrm{lfp}(\lambda x.a \sqcup_A [\![stm]\!]^A (\!|\, b \,|\!)^A x)\big)$$

Of course, in the abstract fixpoint computation for a while-loop, a widening operator ∇_A may be used in place of the lub $\sqcup_A$ for abstractions A that do not satisfy the ascending chain condition or to accelerate convergence in A.

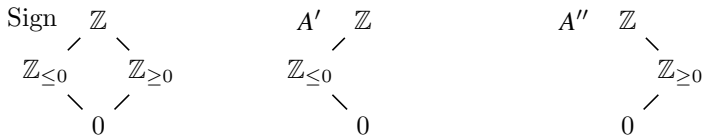
Soundness of the abstract semantics, that is $\alpha([\![P]\!]) \leq_A [\![P]\!]^A$, is a consequence of three main points:

(1) Local soundness of abstract functions is preserved by their composition;
(2) The abstract lub $\sqcup_A$ is a sound approximation of the union $\cup$;
(3) Local soundness implies global fixpoint soundness.

3 Completeness in Program Analysis

Given a program P with n integer variables, in order to simplify the notation, we view a n-tuple of integer values as a store for P so that $\langle \wp(\mathbb{Z}^n), \subseteq \rangle$ plays the role of concrete domain and store (either relational or non-relational) abstractions range in $\mathrm{Abs}(\wp(\mathbb{Z}^n))$. Let us consider the simplest case of one single integer variable. Observe that the trivial abstraction $A^\top = \{\mathbb{Z}\}$, which abstracts each set of integers to the top value $\mathbb{Z}$, is

always obviously complete, meaning that the mapping $\{\mathbb{Z} \mapsto \mathbb{Z}\}$ is both a complete abstract transfer function and a complete abstract program semantics. Let us therefore consider the simplest abstraction which is different from the trivial abstraction $A^\top$: this is a two-point abstraction $\{\mathbb{Z}, K\}$ for some set of integers $K \in \wp(\mathbb{Z})$. To be concrete and simple, let $K = \{0\}$, so that we consider the corresponding abstraction $A^{\mathrm{o}} \triangleq \{\mathbb{Z}, \{0\}\} \in \mathrm{Abs}(\wp(\mathbb{Z}))$. Note that A^{o} is only able to represent precisely the fact that an integer variable has a value equals to 0 and that $A^{\mathrm{o}}(\varnothing) = \{0\}$. Let us observe that A^{o} is not complete for the transfer function $(\!|\, x > 0 \,|\!)$: for example, we have that $A^{\mathrm{o}}\big((\!|\, x > 0 \,|\!)\{-1\}\big) = A^{\mathrm{o}}(\varnothing) = \{0\} \subsetneq \mathbb{Z} = (\!|\, x > 0 \,|\!)^{A^{\mathrm{o}}} A^{\mathrm{o}}(\{-1\})$. Why A^{o} is not complete for the test $x > 0$? As the example highlights, the problem lies in the fact that if S is a set of integers that do not satisfy the test $x > 0$ then the abstraction of $(\!|\, x > 0 \,|\!)S$ boils down to the abstraction of the empty set $\varnothing$ which is $\{0\}$, whereas the only option for A^{o} is to abstract one such set S to $\mathbb{Z}$ and this gives rise to incompleteness. Suitable refinements of the abstraction A^{o} can avoid this incompleteness: one such refined abstraction should be complete for $(\!|\, x > 0 \,|\!)$ while preserving the expressiveness of A^{o}. For example, the Sign abstraction [8] depicted below on the left



turns out to be complete for $(\!|\, x > 0 \,|\!)$ and, clearly, is more precise than A^{o}. Yet, for the specific goal of being complete for $(\!|\, x > 0 \,|\!)$, Sign is too refined, i.e. precise. In fact, the abstraction A' depicted above in the middle is still complete for $(\!|\, x > 0 \,|\!)$, while being simpler than Sign and more precise than A^{o}. What we really need here is a *minimal refinement* of A^{o} which is complete for $(\!|\, x > 0 \,|\!)$. This type of abstraction refinements have been called *complete shell* refinements [13].

3.1 Shells

Recall that abstractions of a common concrete domain D are preordered w.r.t. their relative precision: If $A_1, A_2 \in \mathrm{Abs}(D)$, where $\langle \alpha_i, \gamma_i \rangle$ are the corresponding Galois connections, then $A_1 \trianglelefteq A_2$, i.e. A_1 is a refinement of A_2 and A_2 is a simplification of A_1, when for all $d \in D$, $\gamma_1(\alpha_1(d)) \leq_D \gamma_2(\alpha_2(d))$. Moreover, A_1 and A_2 are equivalent when $A_1 \trianglelefteq A_2$ and $A_2 \trianglelefteq A_1$. By a slight abuse of notation, $\mathrm{Abs}(D)$ denotes the family of abstractions of D up to the above equivalence. It is well known [9] that $\langle \mathrm{Abs}(D), \trianglelefteq \rangle$ is a complete lattice. Given a family of abstract domains $\mathcal{X} \subseteq \mathrm{Abs}(D)$, their lub $\sqcup\mathcal{X}$ is therefore the most precise domain in $\mathrm{Abs}(D)$ which is a simplification of any domain in $\mathcal{X}$.

In a general abstract interpretation framework, Giacobazzi et al. [13] have shown that an abstraction A can be made complete for a family of continuous concrete functions through a minimal refinement of A. Consider a continuous concrete function $f : D \to D$ and an abstraction $A \in \mathrm{Abs}(D)$. Then, A is refined to the complete shell $\mathrm{CShell}_f(A)$ which can be obtained as follows (we use a simplified characterization proved in [2, Theorem 4.1]). We first define a transform $R_f : \wp(D) \to \wp(D)$

as $R_f(X) \triangleq \cup_{a \in X} \max\{d \in D \mid f(d) \leq a\}$, namely $R_f(X)$ collects, for any value $a \in X$, the maximal concrete values d whose f-image is approximated by a. Then, we define $\mathrm{CShell}_f(A) \triangleq \mathrm{Cl}_\wedge(R_f^\omega(\gamma(A)))$, where $\mathrm{Cl}_\wedge$ denotes the closure under glb's $\wedge$ of subsets of D and R_f^ω denotes the ω-closure of R_f, i.e., $R_f^\omega(X) \triangleq \cup_{i \in \mathbb{N}} R_f^i(X)$ (note that $R_f^0(X) = X$). The main result in [13] shows that

$$\mathrm{CShell}_f(A) = \sqcup\{A' \in \mathrm{Abs}(D) \mid A' \trianglelefteq A,\ A' \text{ is complete for } f\}$$

namely, $\mathrm{CShell}_f(A)$ is the least f-complete refinement of A.

Similarly, an abstraction can be also minimally refined in order to be exact for any given set of functions [12]. Given a concrete function $f : D \to D$, any abstraction $A \in \mathrm{Abs}(D)$ can be refined to the exact shell $\mathrm{EShell}_f(A)$ for f as follows: we define $S_f : \wp(D) \to \wp(D)$ as $S_f(X) \triangleq \{f(x) \in D \mid x \in X\}$ and $\mathrm{EShell}_f(A) \triangleq \mathrm{Cl}_\wedge(S_f^\omega(\gamma(A)))$. Thus, $S_f^\omega(\gamma(A))$ is the closure of the abstraction A (more precisely, of its concrete image $\gamma(A)$) under applications of the concrete function f while the exact shell $\mathrm{EShell}_f(A)$ is the closure of $S_f^\omega(\gamma(A))$ under glb's in D. Here, we have that

$$\mathrm{EShell}_f(A) = \sqcup\{A' \in \mathrm{Abs}(D) \mid A' \trianglelefteq A,\ A' \text{ is exact for } f\}.$$

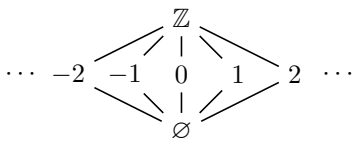
3.2 Signs

The above general solution [13] tells us that in order to make A complete we need to iteratively add to A the maximal concrete values d such that $f(d) \leq a$ for some abstract value $a \in A$ until this process of adding new values to A converges, and at the end this set of concrete values must be closed under glb's. Thus, for the abstract value $\{0\} \in A^0$, we need to consider $\max\{S \in \wp(\mathbb{Z}) \mid (\!|x > 0|\!)S \subseteq \{0\}\}$ and this provides the concrete value $\mathbb{Z}_{\leq 0}$ to join to A^0. In turn, $\max\{S \in \wp(\mathbb{Z}) \mid (\!|x > 0|\!)S \subseteq \mathbb{Z}_{\leq 0}\} = \mathbb{Z}_{\leq 0}$, so that we do not need to further refine A^0. Hence, $\mathrm{Shell}_{(\!|x>0|\!)}(A) = A'$. Likewise, we have that $A'' = \{\mathbb{Z}, \mathbb{Z}_{\geq 0}, 0\}$ (which is depicted above) is the complete shell refinement of A for the transfer function $(\!|x < 0|\!)$. Furthermore, Sign can be obtained as complete shell refinement of A for both transfer functions $(\!|x > 0|\!)$ and $(\!|x < 0|\!)$.

We have therefore analyzed the genesis of a toy abstract domain like Sign from the viewpoint of completeness: Sign is characterized as the minimal refinement of a basic domain $\{\mathbb{Z}, 0\}$ whose abstract transfer functions for the boolean tests $x > 0?$ and $x < 0?$ are complete. In the following, we will show that this completeness-based view on abstraction design can be very useful to understand the genesis of a range of numerical abstractions.

3.3 Constant propagation

Let us consider the standard abstraction CP used in constant propagation analysis [16]:



CP is more precise than the basic abstraction A^{o}, hence it makes sense to ask whether CP can be viewed as a complete shell refinement of A^{o} for some transfer functions. From the viewpoint of its completeness properties, one may observe that CP is clearly complete for the transfer functions $(\!|\, x := x + k \,|\!)$, for any $k \in \mathbb{Z}$. Moreover, we also notice that if an abstraction A is complete for $(\!|\, x := x + 1 \,|\!)$ and $(\!|\, x := x - 1 \,|\!)$ then, since completeness is preserved by composing functions, A is also complete for all the transfer functions $(\!|\, x := x + k \,|\!)$, for all $k \in \mathbb{Z}$, and this allows us to focus on $(\!|\, x := x \pm 1 \,|\!)$ only. Actually, it turns out that completeness for $(\!|\, x := x \pm 1 \,|\!)$ completely characterizes constant propagation CP, in the sense that the complete shell of A^{o} for both $(\!|\, x := x + 1 \,|\!)$ and $(\!|\, x := x - 1 \,|\!)$ is precisely CP. To this aim, it is enough to note that

$$\begin{array}{ll} \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x - 1 \,|\!)S \subseteq \{0\}\} = \{1\} & \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x - 1 \,|\!)S \subseteq \{1\}\} = \{2\} & \cdots \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x + 1 \,|\!)S \subseteq \{0\}\} = \{-1\} & \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x + 1 \,|\!)S \subseteq \{-1\}\} = \{-2\} & \cdots \end{array}$$

and that at the end the closure under intersection adds the empty set $\varnothing$.

3.4 Intervals

Sign is not complete for the transfer function $(\!|\, x := x + 1 \,|\!)$. In fact, notice that

$$\begin{array}{l} \mathrm{Sign}((\!|\, x := x + 1 \,|\!)\{-1\}) = \mathrm{Sign}(\{0\}) = \{0\} \\ (\!|\, x := x + 1 \,|\!)^{\mathrm{Sign}}\,\mathrm{Sign}(\{-1\}) = (\!|\, x := x + 1 \,|\!)^{\mathrm{Sign}}(\mathbb{Z}_{\leq 0}) = \mathbb{Z} \end{array}$$

The well-known domain Int of integer intervals [8] is instead clearly complete for $(\!|\, x := x + 1 \,|\!)$ and $(\!|\, x := x - 1 \,|\!)$. As shown in [13], the complete shell of Sign for $(\!|\, x := x + 1 \,|\!)$ and $(\!|\, x := x - 1 \,|\!)$ turns out to be Int. In fact, we have that:

$$\begin{array}{ll} \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x \pm 1 \,|\!)S \subseteq \mathbb{Z}_{\leq 0}\} = \mathbb{Z}_{\leq \mp 1} & \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x \pm 1 \,|\!)S \subseteq \mathbb{Z}_{\leq -1}\} = \mathbb{Z}_{\leq \mp 2} & \cdots \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x \pm 1 \,|\!)S \subseteq \mathbb{Z}_{\geq 0}\} = \mathbb{Z}_{\geq \mp 1} & \\ \max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x := x \pm 1 \,|\!)S \subseteq \mathbb{Z}_{\geq 1}\} = \mathbb{Z}_{\geq \mp 2} & \cdots \end{array}$$

and clearly the family of sets $\mathbb{Z}_{\leq k}$ and $\mathbb{Z}_{\geq k}$, where k ranges in $\mathbb{Z}$, generate by intersection all the integer intervals in Int.

It is worth noting that, in general, completeness is not preserved under abstraction refinements. In fact, while Sign is complete for $(\!|\, x > 0 \,|\!)$, its complete shell Int for $(\!|\, x := x \pm 1 \,|\!)$ loses this completeness: for example, we have that $\mathrm{Int}((\!|\, x > 0 \,|\!)\{0, 2\}) = [2, 2]$ while $(\!|\, x > 0 \,|\!)^{\mathrm{Int}}\,\mathrm{Int}(\{0, 2\}) = [1, +\infty) \sqcap [0, 2] = [1, 2]$. It is simple to observe that if we try to compensate this lack by the complete shell of Int for $(\!|\, x > k \,|\!)$, for all $k \in \mathbb{Z}$, we end up with the whole concrete domain $\wp(\mathbb{Z})$: this depends on the fact that

$$\max\{S \in \wp(\mathbb{Z}) \mid (\!|\, x > k \,|\!)S \subseteq \mathbb{Z}_{\geq k+2}\} = \mathbb{Z}_{\neq k+1}$$

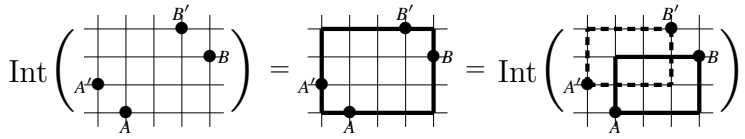
and then by closing under intersections the family of sets $\{\mathbb{Z}_{\neq k}\}_{k \in \mathbb{Z}}$ we get $\wp(\mathbb{Z})$. If instead we consider the complete shell of Int for the specific transfer function $(\!|\, x > 0 \,|\!)$ then Int would be refined to $\mathrm{Int}^{\neq 1} \triangleq \mathrm{Int} \cup \{I \smallsetminus \{1\} \mid I \in \mathrm{Int}\}$, namely $\mathrm{Int}^{\neq 1}$ is able to represent precisely integer intervals with an inner hole $\{1\}$.

3.5 Octagons

Let us consider the program $P \equiv x := 0;\ y := 4;\ \textbf{while}\, x \neq 0\, \textbf{do}\, \{x{-}{-};\ y{+}{+}\}$. According to Miné [17], this is one paradigmatic program showing the need for relational abstract domains. In fact, we have that the interval abstraction Int is not complete for P. Here, $\mathrm{Store} = \wp(\mathbb{Z}^2)$ so that:

$$\mathrm{Int}([\![P]\!]\langle x/\{4\}, y/\{0\}\rangle) = \langle x/[0,0], y/[4,4]\rangle$$
$$[\![P]\!]^{\mathrm{Int}}\,\mathrm{Int}(\langle x/\{4\}, y/\{0\}\rangle) = \langle x/[0,0], y/[0,+\infty)\rangle$$

Of course, the problem of Int with P is that Int is not able to represent precisely the invariant property $x + y = 4$ of the while-loop. Miné [17] then proposes to refine intervals to the so-called octagon abstraction $\mathrm{Oct} \in \mathrm{Abs}(\wp(\mathrm{Store}))$, which is able to express precisely sets of integers like $\{\langle x, y\rangle \in \mathbb{Z}^2 \mid x \pm y = k\}$, where $k \in \mathbb{Z}$. Observe that a set like $S = \{\langle x/0, y/4\rangle, \langle x/2, y/2\rangle, \langle x/4, y/0\rangle\}$ cannot be represented precisely in Oct, since Oct must add $\langle x/1, y/3\rangle$ and $\langle x/3, y/1\rangle$ in the approximation of S. Thus, the observation here is that Oct is exact for the function $\lambda S.S \cup (\!|\, x{-}{-};\ y{+}{+} \,|\!)S$ which is iterated in the fixpoint computation of the while-loop of P. It is worth remarking that Oct is also complete for $\lambda S.S \cup (\!|\, x{-}{-};\ y{+}{+} \,|\!)S$, but this is not the distinguishing feature of Oct compared to Int because Int is already complete for this function, as the following picture suggests:



What we really need is an abstract domain which is able to express precisely the invariant $x + y = 4$ in each iteration step in the fixed point computation of the while-loop of P. We thus refine Int to its exact shell for the function $F \triangleq \lambda S.S \cup (\!|\, x{-}{-};\ y{+}{+} \,|\!)S$ and, dually, for $G \triangleq \lambda S.S \cup (\!|\, x{+}{+};\ y{-}{-} \,|\!)S$. This exact shell can be simply characterized as follows: for a generic point $\langle a, b\rangle \in \wp(\mathbb{Z}^2)$, we have that

$$F^0(\{\langle a, b\rangle\}) = \{\langle a, b\rangle\}$$
$$F^1(\{\langle a, b\rangle\}) = \{\langle a, b\rangle, \langle a-1, b+1\rangle\}$$
$$F^2(\{\langle a, b\rangle\}) = \{\langle a, b\rangle, \langle a-1, b+1\rangle, \langle a-2, b+2\rangle\}$$
$$\ldots$$

so that $F^\omega(\{\langle a, b\rangle\}) = \{\langle x, y\rangle \in \mathbb{Z}^2 \mid x + y = a + b,\ x \leq a,\ y \geq b\}$. Likewise, we have that $G^\omega(\{\langle a, b\rangle\}) = \{\langle x, y\rangle \in \mathbb{Z}^2 \mid x - y = a + b,\ x \geq a,\ y \leq b\}$. Then, by closing under intersections Int, $\{F^\omega(\{\langle a, b\rangle\})\}_{\langle a,b\rangle \in \mathbb{Z}^2}$ and $\{G^\omega(\{\langle a, b\rangle\})\}_{\langle a,b\rangle \in \mathbb{Z}^2}$ we precisely get all the octagons in Oct.

3.6 Polyhedra

The well-known polyhedra abstraction [10] can be characterized as a simple further step of exact refinement of octagons. It is enough to consider a generic program scheme

like

$$Q \equiv x := x_0;\, y := y_0;\, \textbf{while}\,(*)\,\textbf{do}\,\{x := x + k_x;\ y := y + k_y\}$$

Of course, in this case Oct fails to represent precisely the invariant $k_x y - k_y x = k_x y_0 - k_y x_0$ of the while-loop of Q.

It is therefore clear that the exact shell refinement of Oct (or Int) for the family of functions $F = \{\lambda S.S \cup (\!| x := x + k_x;\ y := y + k_y |\!) S\}_{\langle k_x, k_y\rangle \in \mathbb{Z}^2}$ adds to Oct all the linear sets $\{\langle x, y\rangle \in \mathbb{Z}^2 \mid ax + by = c\}$, where a, b, c range in $\mathbb{Z}$, so that the closure under intersections (i.e., logical conjunction) precisely provides the polyhedra abstraction.

4 Completeness in Model Checking

Consider a system model specified as a Kripke structure $\mathcal{K} = (\mathrm{State}, \rightarrow)$. Standard abstract model checking [5] consists in approximating $\mathcal{K}$ by an abstract Kripke structure $\mathcal{A} = (\mathrm{State}^\sharp, \rightarrow^\sharp)$, where the set $\mathrm{State}^\sharp$ of abstract states is defined by a surjective map $h : \mathrm{State} \rightarrow \mathrm{State}^\sharp$ that joins together indistinguishable concrete states. Thus, $\mathrm{State}^\sharp$ determines a partition of State and vice versa any partition of State can be viewed as a set of abstract states. It is simple to view state partitions as abstractions [19]. Any state partition $P \in \mathrm{Part}(\mathrm{State})$ can be viewed as an abstraction $P^a \in \mathrm{Abs}(\wp(\mathrm{State})_\subseteq)$ that approximates any set S of states by the minimal cover of S in the partition P, i.e., $\cup\{B \in P \mid B \cap S \neq \varnothing\}$. Hence, $P^a \triangleq \{S \subseteq \mathrm{State} \mid \exists\{B_i\}_{i\in I} \subseteq P.\ S = \cup_{i\in I} B_i\}$, ordered by subset inclusion, is viewed as an abstraction in $\mathrm{Abs}(\wp(\mathrm{State}))$ that encodes the state partition P. Furthermore, if $A \in \mathrm{Abs}(\wp(\mathrm{State}))$ is any abstraction then A is the encoding of some state partition P, i.e. $A = P^a$, iff A is exact for the negation function $\neg : \wp(\mathrm{State}) \rightarrow \wp(\mathrm{State})$ such that $\neg S = \mathrm{State} \smallsetminus S$. This embedding of the lattice of partitions $\mathrm{Part}(\mathrm{State})$ into the lattice of abstractions $\mathrm{Abs}(\wp(\mathrm{State}))$ allows us to reason on the completeness and exactness properties of partitions, and hence of abstract Kripke structures.

4.1 Strong Preservation

Given some temporal specification language $\mathfrak{L}$ — like CTL, ACTL, μ-calculus, etc. — and a corresponding interpretation of its formulae on the states of a Kripke structure $\mathcal{K}$, an abstract Kripke structure $\mathcal{A}$ *preserves* $\mathfrak{L}$ when for any $\varphi \in \mathfrak{L}$ and $s \in \mathrm{State}$, $h(s) \models^{\mathcal{A}} \varphi \Rightarrow s \models^{\mathcal{K}} \varphi$, while $\mathcal{A}$ *strongly preserves* $\mathfrak{L}$ when $h(s) \models^{\mathcal{A}} \varphi \Leftrightarrow s \models^{\mathcal{K}} \varphi$ holds.

Given a language $\mathfrak{L}$ and a Kripke structure $\mathcal{K}$, a well-known key problem is to compute the smallest abstract state space $\mathrm{State}^\sharp_{\mathfrak{L}}$, when this exists, such that one can define an abstract Kripke structure $\mathcal{A}_{\mathfrak{L}} = (\mathrm{State}^\sharp_{\mathfrak{L}}, \rightarrow^\sharp)$ that strongly preserves $\mathfrak{L}$. This problem admits solution for a number of well-known temporal languages like Hennessy-Milner logic HML, CTL, ACTL and $\mathrm{CTL\text{-}X}$ (i.e. CTL without the next-time operator X). A number of algorithms for solving this problem exist, like those by Paige and Tarjan [18] for HML and CTL, by Henzinger et al. [15] and Ranzato and Tapparo [21] for ACTL, and Groote and Vaandrager [14] for $\mathrm{CTL\text{-}X}$. These are *coarsest partition refinement* algorithms: given a language $\mathfrak{L}$ and an initial state partition P, which is determined by a state labeling in $\mathcal{K}$, these algorithms can be viewed as computing the

coarsest partition $P_{\mathfrak{L}}$ that refines P and induce an abstract Kripke structure that strongly preserves $\mathfrak{L}$. It is worth remarking that these algorithms have been designed for computing some well-known behavioural equivalences used in process algebra like bisimulation (for CTL), simulation (for ACTL) and stuttering bisimulation (for CTL-X). Our abstract interpretation approach allows us to provide a generalized view of these partition refinement algorithms based on exactness properties.

4.2 Bisimulation

Given a partition $P \in \mathrm{Part}(\mathrm{State})$ and a state s, $P(s)$ denotes the block of P that contains s. Then, P is a bisimulation on a Kripke structure $\mathcal{K}$ when $P(s) = P(t)$ and $s \to s'$ imply that there exists some state t' such that $t \to t'$ and $P(s') = P(t')$. It is well known [5] that P is a bisimulation iff the abstract Kripke structure $\mathcal{A}_P = \langle P, \to^{\exists} \rangle$ strongly preserves HML (or, equivalently, CTL), where $B_1 \to^{\exists} B_2$ iff there exist $s_i \in B_i$ such that $s_1 \to s_2$. Moreover, the coarsest partition in $\mathrm{Part}(\mathrm{State})$ which is a bisimulation on $\mathcal{K}$ exists and is called bisimulation equivalence.

Bisimulation for P can be expressed as an exacteness property of the abstraction P^a [19]. The standard predecessor operator $\mathrm{pre} : \wp(\mathrm{State}) \to \wp(\mathrm{State})$ on $\mathcal{K}$ is defined as $\mathrm{pre}(T) \triangleq \{s \in \mathrm{State} \mid s \to t,\ t \in T\}$ and is here considered as a concrete function. Hence, it turns out that P is a bisimulation iff P^a is exact for pre. As a consequence, any partition refinement algorithm for computing bisimulation can be characterized as an exact shell abstraction refinement for: (1) the negation operator $\neg$, in order to ensure that the abstraction is indeed a partition and (2) the predecessor operator pre, in order to ensure that the partition is a bisimulation. In particular, the Paige-Tarjan bisimulation algorithm [18] can be viewed as an efficient implementation of this exact shell computation.

4.3 Simulation

Given a preorder relation $R \in \mathrm{PreOrd}(\mathrm{State})$ on states, $R(s)$ denotes $\{t \in \mathrm{State} \mid (s,t) \in R\}$ while $P_R \triangleq R \cap R^{-1} \in \mathrm{Part}(\mathrm{State})$ denotes the symmetric reduction of R, which being an equivalence relation can be viewed as a state partition.

A preorder R is a simulation on a Kripke structure $\mathcal{K}$ if $t \in R(s)$ and $s \to s'$ imply that there exists some state t' such that $t \to t'$ and $t' \in R(s')$. It is well known [5] that a preorder R is a simulation iff the abstract Kripke structure $\mathcal{A}_R = \langle P_R, \to^{\exists} \rangle$ strongly preserves ACTL and that the largest preorder in $\mathrm{PreOrd}(\mathrm{State})$ which is a simulation on $\mathcal{K}$ exists and is called simulation preorder.

The characterization of simulation as an exactness property follows the lines of bisimulation [19]. Similarly to partitions, a preorder relation R can be viewed as an abstraction $R^a \in \mathrm{Abs}(\wp(\mathrm{State}))$ that approximates any set $S \subseteq \mathrm{State}$ with the image of S through R, i.e. $\cup_{s \in S} R(s)$. Thus, $R^a \triangleq \{S \subseteq \mathrm{State} \mid \exists S \subseteq \mathrm{State}.\ S = \cup_{s \in S} R(s)\}$, ordered by subset inclusion, is viewed as an abstraction in $\mathrm{Abs}(\wp(\mathrm{State}))$ that encodes the preorder relation R. On the other hand, an abstraction $A \in \mathrm{Abs}(\wp(\mathrm{State}))$ is the encoding of some state preorder R, i.e. $A = R^a$, iff A is exact for the union $\cup : \wp(\mathrm{State})^2 \to \wp(\mathrm{State})$. We notice that A is exact for the union iff A is a so-called disjunctive abstraction, so that the lattice of preorder relations $\mathrm{PreOrd}(\mathrm{State})$ can be

embedded into $\mathrm{Abs}(\wp(\mathrm{State}))$ as the sublattice of disjunctive abstractions. This allows us to get the following characterization: a preorder R is a simulation iff R^a is exact for pre. Here, we obtain that any algorithm for computing the simulation preorder can be characterized as an exact shell abstraction refinement for: (1) the union $\cup$, in order to ensure that the abstraction represents a preorder and (2) the predecessor operator pre, in order to ensure that this preorder is a simulation. In particular, the simulation algorithm with the best time complexity [21] has been designed as an efficient implementation of this exact shell computation.

4.4 Stuttering Bisimulation and Simulation

Stuttering bisimulation and simulation relations characterize temporal logics which do not feature the next-time connective X. Let us focus on stuttering simulation. A preorder relation $R \in \mathrm{PreOrd}(\Sigma)$ is a stuttering simulation on $\mathcal{K}$ if $t \in R(s)$ and $s \to s'$ imply that there exist states $t_0, ..., t_k$, with $k \geq 0$, such that: (i) $t_0 = t$, (ii) for all $i \in [0, k[$, $t_i \to t_{i+1}$ and $t_i \in R(s)$, (iii) $t_k \in R(s)$. It turns out that stuttering simulation can be characterized as an exactness property [19]. Following [14], consider the binary stuttering operator $\mathrm{pos} : \wp(\mathrm{State}) \times \wp(\mathrm{State}) \to \wp(\mathrm{State})$ which is defined as follows:

$$\mathrm{pos}(S, T) \triangleq$$
$$\{s \in S \mid \exists\, k \geq 0.\, \exists\, s_0, ..., s_k.\ s_0 = s\ \&\ \forall\, i \in [0, k).\, s_i \in S,\ s_i \to s_{i+1}\ \&\ s_k \in T\}.$$

Hence, it turns out that a preorder R is a stuttering simulation iff R^a is exact for pos. As shown in [20], this allows us to design an efficient algorithm for computing the stuttering simulation preorder as an exact shell abstraction refinement for the union $\cup$ and the stuttering operator pos.

4.5 Probabilistic Bisimulation and Simulation

The main behavioral relations between concurrent systems, like simulation and bisimulation, have been studied in probabilistic extensions of reactive systems like Markov chains and probabilistic automata. As recently shown in [11], we mention that bisimulation and simulation relations on probabilistic transition systems can still be characterized as exactness properties in abstract interpretation and as a byproduct this allows to design efficient algorithms that compute these behavioral relations as exact shell refinements.

5 Conclusion

We have shown how completeness and exactness properties of abstractions play an ubiquitous role in static analysis by exhibiting an array of examples ranging from abstract interpretation-based program analysis to abstract model checking. We are convinced that it is often rewarding to look at scenarios based on some form of semantic approximation under the light of completeness/exactness: this can be useful both to understand the deep logic of the approximation and to profit of the beneficial toolkit that completeness brings, like complete and exact shell refinements.

Acknowledgements. I would like to thank Roberto Giacobazzi for his constant and passionate brainstorming on completeness and exactness: Roberto and I are both persuaded that completeness can be spotted really everywhere.

References

1. C. Aarts, R. Backhouse, E. Boiten, H. Doornbos, N. van Gasteren, R. van Geldrop, P. Hoogendijk, E. Voermans, J. van der Woude. Fixed-point calculus. *Inform. Process. Lett.*, 53(3): 131-136, 1995.
2. G. Amato and F. Scozzari. Observational completeness on abstract interpretation. *Fundamenta Informaticae*, 47(12):1533-1560, 2011.
3. K.R. Apt and G.D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724-767, 1986.
4. J.W. de Bakker, J.-J.C. Meyer and J. Zucker. On infinite computations in denotational semantics. *Theoret. Comput. Sci.*, 26(1-2):53-82, 1983.
5. E.M. Clarke, O. Grumberg and D.A. Peled. *Model checking*. The MIT Press, 1999.
6. P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. Ph.D. dissertation, Université Scientifique et Médicale de Grenoble, Grenoble, France, 1978.
7. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1-2):47-103, 2002.
8. P. Cousot and R. Cousot. 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. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM POPL*, pp. 269–282, 1979.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. 5th ACM POPL*, pp. 84-97, 1978.
11. S. Crafa and F. Ranzato Bisimulation and simulation algorithms on probabilistic transition systems by abstract interpretation. *Formal Methods in System Design*, 40(3):356-376, 2012.
12. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model checking. In *Proc. 8th SAS*, LNCS 2126, pp. 356-373, 2001.
13. R. Giacobazzi, F. Ranzato and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361-416, 2000.
14. J.F. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. ICALP'90*, LNCS 443, pp. 626-638, Springer, 1990.
15. M.R. Henzinger, T.A. Henzinger and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, pp. 453–462, IEEE Press, 1995.
16. G. Kildall. A unified approach to global program optimization. In *Proc. 1st ACM POPL*, pp. 194–206, 1973.
17. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31-100, 2006.
18. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987
19. F. Ranzato and F. Tapparo. Generalized strong preservation by abstract interpretation. *J. Logic and Computation*, 17(1):157-197, 2007.
20. F. Ranzato and F. Tapparo. Computing stuttering simulations. In *Proc. 20th Int. Conf. on Concurrency Theory (CONCUR'09)*, LNCS vol. 5710, pp. 542-556, Springer, 2009.
21. F. Ranzato and F. Tapparo. An efficient simulation algorithm based on abstract interpretation. *Information and Computation*, 208(1):1-22, 2010.