Logical Abstract Domains and Interpretations

Patrick Cousot, Radhia Cousot, Laurent Mauborgne

Logical Abstract Domains and Interpretations

Patrick Cousot [2,3], Radhia Cousot [3,1], and Laurent Mauborgne [3,4]

[1] Centre National de la Recherche Scientifique, Paris
[2] Courant Institute of Mathematical Sciences, New York University
[3] École Normale Supérieure, Paris
[4] Instituto Madrileño de Estudios Avanzados, Madrid

Abstract. We give semantic foundations to abstract domains consisting in first order logic formulæ in a theory, as used in verification tools or methods using SMT-solvers or theorem provers. We exhibit conditions for a sound usage of such methods with respect to multi-interpreted semantics and extend their usage to automatic invariant generation by abstract interpretation.

1 Introduction

Hoare's axiomatic logic [35,13] can be formalized as defining a program semantics $C[\![\mathtt{P}]\!]$ which is the set of inductive invariants $C[\![\mathtt{P}]\!] \triangleq \left\{ I \in A \,\middle|\, F[\![\mathtt{P}]\!](I) \sqsubseteq I \right\}$ where $F[\![\mathtt{P}]\!] \in A \to A$ is the transformer of program P in an abstract domain $\langle A, \sqsubseteq, \bot, \sqcup \rangle$, $\sqsubseteq$ is a pre-order, $\bot$ is the infimum, and the join $\sqcup$, if any, is the least upper bound (or an over-approximation) up to the pre-order equivalence. Program verification, consists in proving that a program specification $S \in A$ is implied by a program inductive invariant, that is $\exists I \in C[\![\mathtt{P}]\!] : I \sqsubseteq S$.

To be of interest, the semantics $C[\![\mathtt{P}]\!]$ must be assumed to be non-empty, which can be ensured by additional hypotheses. For example, the existence of a supremum $\top \in A$ ensures $\top \in C[\![\mathtt{P}]\!]$. A more interesting particular case is when $F[\![\mathtt{P}]\!] \in A \stackrel{\nearrow}{\to} A$ is increasing and $\langle A, \sqsubseteq, \bot, \sqcup \rangle$ is a cpo (or a complete lattice) in which case the $\sqsubseteq$-least fixpoint $\mathbf{lfp}^{\sqsubseteq} F[\![\mathtt{P}]\!]$ does exist, up to the pre-order equivalence, e.g. by [46], and is the strongest invariant. Besides soundness, the existence of this strongest invariant ensures the completeness of the verification method. Another interesting case is when $F[\![\mathtt{P}]\!]$ is increasing or extensive so that the iterates $F[\![P]\!]^0(\bot) \triangleq \bot$, $F[\![P]\!]^{\lambda+1}(\bot) \triangleq F[\![P]\!](F[\![P]\!]^\lambda(\bot))$ and $F[\![P]\!]^\lambda(\bot) \triangleq \bigsqcup_{\beta<\lambda} F[\![P]\!]^\beta(\bot)$ for limit ordinals λ (or $\forall \beta < \lambda : F[\![P]\!]^\lambda(\bot) \sqsupseteq F[\![P]\!]^\beta(\bot)$ in absence of join) do converge. The limit $F[\![P]\!]^\epsilon(\bot)$ is necessarily a fixpoint of $F[\![P]\!]$, which is therefore an inductive invariant, but maybe not the strongest one. When $F[\![P]\!]$ is continuous, we have $\epsilon = \omega$, the first infinite ordinal [19].

Automatic program verification can be categorized as follows.

(i) Deductive methods exploit the Floyd/Naur/Hoare proof method [13] that consists in guessing an inductive invariant $I \in A$ and proving that $F[\![\mathtt{P}]\!](I) \sqsubseteq I \wedge I \sqsubseteq S$. The end-user provides the inductive invariant $I \in A$ and a deductive system provides the correctness proof;
(ii) Fixpoint iterates generalization [12] consists in finding an over-approximation I^λ of the inductive definition of the iterates $F[\![P]\!]^\lambda(\bot)$ of $F[\![P]\!]$, machine-check that

$F[\![P]\!](I^\lambda) \sqsubseteq I^{\lambda+1}$ and $I^\lambda \sqsupseteq \bigsqcup_{\beta<\lambda} I^\beta$ for limit ordinals (proving that $\forall \lambda \geqslant 0 : F[\![P]\!]^\lambda(\bot) \sqsubseteq I^\lambda$ by recurrence and $F[\![P]\!]$ is increasing or extensive). For the limit I^ε, we have $F[\![P]\!]^\epsilon(\bot) \sqsubseteq I^\varepsilon$ so that $I^\varepsilon \sqsubseteq S$ implies $\exists\, I \in C[\![\mathtt{P}]\!] : I \sqsubseteq S$;

(iii) Model checking [8] considers the case when A is a finite complete lattice so that $\sqsubseteq$, $F[\![\mathtt{P}]\!]$ which is assumed to be increasing, and $\mathbf{lfp}^\sqsubseteq F[\![\mathtt{P}]\!]$ are all computable;

(iv) Bounded model checking [7] aims at proving that the specification is not satisfied by proving $F[\![\mathtt{P}]\!]^k(\bot) \not\sqsubseteq S$ which implies $\mathbf{lfp}^\sqsubseteq F[\![\mathtt{P}]\!] \not\sqsubseteq S$, e.g. by [46];

(v) Static analysis by abstract interpretation [18,20] consists in effectively computing an abstract invariant $I^\sharp$ which concretization $\gamma(I^\sharp)$ is automatically proved to satisfy $F[\![\mathtt{P}]\!](\gamma(I^\sharp)) \sqsubseteq \gamma(I^\sharp) \wedge \gamma(I^\sharp) \sqsubseteq S$. The automatic computation of $I^\sharp$ is based on (ii) in the abstract, using an abstract domain satisfying the ascending chain condition or else a widening to inductively over-approximate the concrete iterates.

By undecidability, none of these methods can be simultaneously automatic, terminating, sound, and complete on the interpreted semantics of all programs of a non-trivial programming language. Moreover all methods (i)—(iv) restrict the expressible runtime properties hence involve some form of abstraction (v).

Of particular interest is the case of program properties $A \subseteq \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ expressed as first-order formulæ on variables $\mathbb{x}$, function symbols $\mathbb{f}$, and predicates symbols $\mathbb{p}$ and $\sqsubseteq$ is logical implication $\Rightarrow$. Recent progress in SMT solvers and more generally theorem provers [4] has been exploited in deductive methods and bounded model checking on $\langle A,\ \Rightarrow,\ \mathtt{false}\rangle$ or combinations of $A_i \subseteq \mathbb{F}(\mathbb{x}, \mathbb{f}_i, \mathbb{p}_i)$, $i = 1, \ldots, n$ exploiting the Nelson-Oppen procedure [41] for combining decidable theories. We study abstract interpretation-based static analysis restricted to logical abstract domains $A \subseteq \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$, the semantic foundations, and the necessary generalization from invariant verification to invariant generation [5].

2 Terminology on First-Order Logics, Theories, Interpretations and Models

2.1 First-order logics

We define $\mathcal{B} \triangleq \{\mathit{false}, \mathit{true}\}$ to be the Booleans. The set $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ of first-order formulæ on a signature $\langle \mathbb{f},\ \mathbb{p}\rangle$ (where $\mathbb{f}$ are the function symbols, and $\mathbb{p}$ the predicate symbols such that $\mathbb{f} \cap \mathbb{p} = \emptyset$) and variables $\mathbb{x}$, is defined as:

$\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots \in \mathbb{x}$	variables
$\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots \in \mathbb{f}^0$	constants, $\mathbb{f} \triangleq \bigcup_{n\geqslant 0} \mathbb{f}^n$
$\mathtt{f}, \mathtt{g}, \mathtt{h}, \ldots \in \mathbb{f}^n$	function symbols of arity $n \geqslant 1$
$t \in \mathbb{T}(\mathbb{x}, \mathbb{f})$ $\quad t ::= \mathtt{x} \mid \mathtt{c} \mid \mathtt{f}(t_1, \ldots, t_n)$	terms
$\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots \in \mathbb{p}^n, \quad \mathbb{p} \triangleq \bigcup_{n\geqslant 0} \mathbb{p}^n$	predicate symbols of arity $n \geqslant 0$, $\mathbb{p}^0 \triangleq \{\mathtt{ff}, \mathtt{tt}\}$
$a \in \mathbb{A}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ $\quad a ::= \mathtt{ff} \mid \mathtt{p}(t_1, \ldots, t_n) \mid \neg a$	atomic formulæ

[5] For example [4] is mainly on invariant verification while Ch. 12 on invariant generation by abstract interpretation is unrelated to the previous chapters using first-order logic theories.

$$
\begin{array}{lll}
e \in \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \triangleq \mathbb{T}(\mathbb{x}, \mathbb{f}) \cup \mathbb{A}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & & \text{program expressions} \\
\varphi \in \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \varphi ::= a \mid \varphi \wedge \varphi & \text{clauses in simple conjunctive normal form} \\
\Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \Psi ::= a \mid \neg\Psi \mid \Psi \wedge \Psi \mid \exists \mathtt{x} : \Psi & \text{quantified first-order formulæ}
\end{array}
$$

In first order logics with equality, there is a distinguished predicate $= (t_1, t_2)$ which we write $t_1 = t_2$.

2.2 Theories

The set $\vec{\mathtt{x}}_\Psi$ of free variables of a formula Ψ is defined inductively as the set of variables in the formula which are not in the scope of an existential quantifier. A *sentence* of $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ is a formula with no free variable. A *theory* is a set of sentences [6] (called the *theorems* of the theory) and a signature, which should contain at least all the predicates and function symbols that appear in the theorems. The *language* of a theory is the set of quantified first-order formulæ that contain no predicate or function symbol outside of the signature of the theory.

The idea of theories is to restrict the possible meanings of functions and predicates in order to reason under these hypotheses. The meanings which are allowed are the meanings which make the sentences of the theory true.

2.3 Interpretations

This is better explained with the notion of interpretation of formulæ: An *interpretation* I for a signature $\langle \mathbb{f},\ \mathbb{p} \rangle$ is a couple $\langle I_\mathcal{V},\ I_\gamma \rangle$ such that $I_\mathcal{V}$ is a non-empty set of values, $\forall \mathtt{c} \in \mathbb{f}^0 : I_\gamma(\mathtt{c}) \in I_\mathcal{V}$, $\forall n \geqslant 1 : \forall \mathtt{f} \in \mathbb{f}^n : I_\gamma(\mathtt{f}) \in I_\mathcal{V}^n \to I_\mathcal{V}$ and $\forall n \geqslant 0 : \forall \mathtt{p} \in \mathbb{p}^n : I_\gamma(\mathtt{p}) \in I_\mathcal{V}^n \to \mathcal{B}$. Let $\mathfrak{I}$ be the class of all such interpretations I. In a given interpretation $I \in \mathfrak{I}$, an environment [6] is a function from variables to values

$$\eta \in \mathcal{R}_I \triangleq \mathbb{x} \to I_\mathcal{V} \quad \text{environments}$$

An interpretation I and an environment η satisfy a formula Ψ, written $I \models_\eta \Psi$, in the following way:

$$
\begin{array}{rclcrcl}
I \models_\eta a & \triangleq & [\![a]\!]_I \eta & \qquad & I \models_\eta \Psi \wedge \Psi' & \triangleq & (I \models_\eta \Psi) \wedge (I \models_\eta \Psi') \\
I \models_\eta \neg\Psi & \triangleq & \neg(I \models_\eta \Psi) & & I \models_\eta \exists \mathtt{x} : \Psi & \triangleq & \exists v \in I_\mathcal{V} :\ I \models_{\eta[\mathtt{x} \leftarrow v]} \Psi \ [7]
\end{array}
$$

where the value $[\![a]\!]_I \eta \in \mathcal{B}$ of an atomic formula $a \in \mathbb{A}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ in environment $\eta \in \mathcal{R}_I$ is

$$
\begin{array}{rcll}
[\![\mathtt{ff}]\!]_I \eta & \triangleq & \mathit{false} & \\
[\![\mathtt{p}(t_1, \ldots, t_n)]\!]_I \eta & \triangleq & I_\gamma(\mathtt{p})([\![t_1]\!]_I \eta, \ldots, [\![t_n]\!]_I \eta), & \text{where} \quad I_\gamma(\mathtt{p}) \in I_\mathcal{V}^n \to \mathcal{B} \\
[\![\neg a]\!]_I \eta & \triangleq & \neg [\![a]\!]_I \eta, & \text{where} \quad \neg \mathit{true} = \mathit{false},\ \neg \mathit{false} = \mathit{true}
\end{array}
$$

and the value $[\![t]\!]_I \eta \in I_\mathcal{V}$ of the term $t \in \mathbb{T}(\mathbb{x}, \mathbb{f})$ in environment $\eta \in \mathcal{R}_I$ is

[6] Environments are also called variable assignments, valuations, etc. For programming languages, environments may also contain the program counter, stack, etc.

[7] $\eta[\mathtt{x} \leftarrow v]$ is the assignment of v to $\mathtt{x}$ in η.

$$
\begin{array}{rcll}
[\![\mathtt{x}]\!]_I\eta & \triangleq & \eta(\mathtt{x}) & \\
[\![\mathtt{c}]\!]_I\eta & \triangleq & I_\gamma(\mathtt{c}), & \text{where} \quad I_\gamma(\mathtt{c}) \in I_\mathcal{V} \\
[\![\mathtt{f}(t_1,\ldots,t_n)]\!]_I\eta & \triangleq & I_\gamma(\mathtt{f})([\![t_1]\!]_I\eta,\ldots,[\![t_n]\!]_I\eta), & \text{where} \quad I_\gamma(\mathtt{f}) \in {I_\mathcal{V}}^n \to I_\mathcal{V},\ n \geqslant 1
\end{array}
$$

In addition, in a first-order logic with equality the interpretation of equality is always

$$
I \models_\eta t_1 = t_2 \quad \triangleq \quad [\![t_1]\!]_I\eta =_I [\![t_2]\!]_I\eta
$$

where $=_I$ is the unique reflexive, symmetric, antisymmetric, and transitive relation on $I_\mathcal{V}$.

2.4 Models

An interpretation $I \in \mathfrak{I}$ is said to be a *model* of Ψ when $\exists\eta : I \models_\eta \Psi$ (i.e. I makes Ψ true). An interpretation is a *model* of a theory iff it is a model of all the theorems of the theory (i.e. makes true all theorems of the theory). The class of all models of a theory $\mathcal{T}$ is

$$
\begin{aligned}
\mathfrak{M}(\mathcal{T}) &\triangleq \{I \in \mathfrak{I} \mid \forall \Psi \in \mathcal{T} : \exists\eta : I \models_\eta \Psi\} \\
&= \{I \in \mathfrak{I} \mid \forall \Psi \in \mathcal{T} : \forall\eta : I \models_\eta \Psi\}
\end{aligned}
$$

since if Ψ is a sentence and if there is a I and a η such that $I \models_\eta \Psi$, then for all η', $I \models_{\eta'} \Psi$.

Quite often, the set of sentences of a theory is not defined extensively, but using a (generally finite) set of axioms which generate the set of theorems of the theory by implication. A theory is said to be *deductive* iff it is closed by deduction, that is all theorems that are true on all models of the theory are in the theory.

The theory of an interpretation I is the set of sentences Ψ such that I is a model of Ψ. Such a theory is trivially deductive and satisfiable (i.e. has at least one model).

2.5 Satisfiability and validity (modulo interpretations and theory)

A formula Ψ is *satisfiable* (with respect to the class of interpretations $\mathfrak{I}$ defined in Sect. **2.3**) if there exists an interpretation I and an environment η that make the formula true ($\mathsf{satisfiable}(\Psi) \triangleq \exists I \in \mathfrak{I} : \exists\eta : I \models_\eta \Psi$). A formula is *valid* if all such interpretations make the formula true ($\mathsf{valid}(\Psi) \triangleq \forall I \in \mathfrak{I} : \forall\eta : I \models_\eta \Psi$). The negations of the concepts are *unsatisfiability* ($\neg\mathsf{satisfiable}(\Psi) = \forall I \in \mathfrak{I} : \forall\eta : I \models_\eta \neg\Psi$) and *invalidity* ($\neg\mathsf{valid}(\Psi) = \exists I \in \mathfrak{I} : \exists\eta : I \models_\eta \neg\Psi$). So Ψ is satisfiable iff $\neg\Psi$ is invalid and Ψ is valid iff $\neg\Psi$ is unsatisfiable.

These notions can be put in perspective in *satisfiability and validity modulo interpretations* $\mathcal{I} \in \wp(\mathfrak{I})$, where we only consider interpretations $I \in \mathcal{I}$. So $\mathsf{satisfiable}_\mathcal{I}(\Psi) \triangleq \exists I \in \mathcal{I} : \exists\eta : I \models_\eta \Psi$ and $\mathsf{valid}_\mathcal{I}(\Psi) \triangleq \forall I \in \mathcal{I} : \forall\eta : I \models_\eta \Psi$ (also denoted $\mathcal{I} \models \Psi$).

The case $\mathcal{I} = \mathfrak{M}(\mathcal{T})$ corresponds to *satisfiability and validity modulo a theory* $\mathcal{T}$, where we only consider interpretations $I \in \mathfrak{M}(\mathcal{T})$ that are models of the theory (i.e. make true all theorems of the theory). So $\mathsf{satisfiable}_\mathcal{T}(\Psi) \triangleq \mathsf{satisfiable}_{\mathfrak{M}(\mathcal{T})}(\Psi) = \exists I \in$

$\mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi$ and $\mathsf{valid}_\mathcal{T}(\Psi) \triangleq \mathsf{valid}_{\mathfrak{M}(\mathcal{T})}(\Psi) = \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \Psi$ (also denoted $\mathcal{T} \models \Psi$).

The four concepts can be extended to theories: a theory is satisfiable [8] (valid) if one (all) of the interpretations is a (are) model(s) of the theory i.e. $\mathfrak{M}(\mathcal{T}) \neq \emptyset$ (resp. $\mathfrak{M}(\mathcal{T}) = \mathfrak{J}$), and a theory is unsatisfiable (invalid) if all (one) of the interpretations make(s) each of the theorems of the theory false.

2.6 Decidable theories

A theory $\mathcal{T}$ is *decidable* iff there is an algorithm $\mathsf{decide}_\mathcal{T} \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathcal{B}$ that can decide in finite time if a given formula is in the theory or not.

Decidable theories provide approximations for the *satisfiability problem*: a formula Ψ is satisfiable iff there is an interpretation I and an environment η such that $I \models_\eta \Psi$ is true ($\mathsf{satisfiable}(\Psi) \triangleq \exists I \in \mathfrak{J} : \exists \eta : I \models_\eta \Psi$). So a formula Ψ with free variables $\vec{\mathbf{x}}_\Psi$ is satisfiable iff the sentence $\exists \vec{\mathbf{x}}_\Psi : \Psi$ obtained from the formula by existentially quantifying the free variables is satisfiable. So if we know that this sentence is in a satisfiable theory, then the original formula is also satisfiable and, in addition, we know that it is satisfiable for all models of that theory.

$$\mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_\Psi : \Psi) \Rightarrow \mathsf{satisfiable}_\mathcal{T}(\Psi) \qquad \text{(when } \mathcal{T} \text{ is decidable and satisfiable)}$$

Proof.

$$\begin{array}{lll}
& \mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_\Psi : \Psi) & \\
\Leftrightarrow & (\exists \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T} & \wr \text{def. decision procedure} \wr \\
\Rightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \exists \vec{\mathbf{x}}_\Psi : \Psi & \\
& & \wr \text{def. } \mathfrak{M}(\mathcal{T}) \triangleq \{I \in \mathfrak{J} \mid \forall \Psi' \in \mathcal{T} : \exists \eta' : I \models_{\eta'} \Psi'\} \wr \\
\Leftrightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi & \wr \text{def. } I \models_\eta \exists \mathbf{x} : \Psi \text{ in Sect. } \mathbf{2.3} \wr \\
\Rightarrow & \exists I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi & \wr \mathcal{T} \text{ is satisfiable so } \mathfrak{M}(\mathcal{T}) \neq \emptyset \wr \\
\Leftrightarrow & \mathsf{satisfiable}_\mathcal{T}(\Psi) & \wr \text{def. } \mathsf{satisfiable}_\mathcal{T}(\Psi) \triangleq \exists I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi \wr \quad \square
\end{array}$$

So the problem of satisfiability modulo a theory $\mathcal{T}$ can be approximated by decidability in $\mathcal{T}$ in the sense that if the decision is *true* then the formula is satisfiable, otherwise we don't know in general.

The same result holds for validity:

$$\mathsf{decide}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi : \Psi) \Rightarrow \mathsf{valid}_\mathcal{T}(\Psi) \qquad \text{(when } \mathcal{T} \text{ is decidable)}$$

Proof.

$$\begin{array}{lll}
& \mathsf{decide}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi : \Psi) & \\
\Leftrightarrow & (\forall \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T} & \wr \text{def. decision procedure} \wr
\end{array}$$

[8] Model theorists often use "consistent" as a synonym for "satisfiable".

$$
\begin{aligned}
\Rightarrow\quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi \\
& \qquad\qquad \wr \text{def. } \mathfrak{M}(\mathcal{T}) \triangleq \{I \in \mathfrak{I} \mid \forall \Psi' \in \mathcal{T} : \exists \eta' : I \models_{\eta'} \Psi'\} \wr \\
\Leftrightarrow\quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi \\
& \qquad \wr \text{since } \forall \vec{\mathbf{x}}_\Psi : \Psi \text{ has no free variable so } I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi \text{ does not depend on } \eta \wr \\
\Leftrightarrow\quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \Psi \qquad \wr \text{def. } I \models_\eta \forall \mathbf{x} : \Psi \text{ in Sect. } \mathbf{2.3} \wr \\
\Leftrightarrow\quad & \mathsf{valid}_\mathcal{T}(\Psi) \qquad \wr \mathsf{valid}_\mathcal{T}(\Psi) \triangleq \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \Psi \wr \quad \square
\end{aligned}
$$

It is possible to obtain implications in the other direction so that we solve exactly the validity or satisfiability problem, when the theory is deductive.

$$
\mathsf{valid}_\mathcal{T}(\Psi) \Leftrightarrow \mathsf{decide}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi : \Psi) \qquad \text{(when } \mathcal{T} \text{ is decidable and deductive)}
$$

Proof. $\mathcal{T}$ is deductive, hence all valid sentences are theorems of the theory, so if $\mathsf{valid}_\mathcal{T}(\Psi)$ then $\forall \vec{\mathbf{x}}_\Psi : \Psi$ is a valid sentence of $\mathcal{T}$ and so it is in $\mathcal{T}$. $\square$

From that, we can obtain satisfiability of any formula:

$$
\mathsf{satisfiable}_\mathcal{T}(\Psi) \Leftrightarrow \neg\,(\mathsf{decide}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi : \neg\Psi)) \quad \text{(when } \mathcal{T} \text{ is decidable and deductive)}
$$

Proof.

$$
\begin{aligned}
& \mathsf{satisfiable}_\mathcal{T}(\Psi) \\
\Leftrightarrow\quad & \neg\,(\mathsf{valid}_\mathcal{T}(\neg\Psi)) \qquad \wr \text{def. } \mathsf{satisfiable}_\mathcal{T} \text{ and } \mathsf{valid}_\mathcal{T} \text{ in Sect. } \mathbf{2.5} \wr \\
\Leftrightarrow\quad & \neg\,(\mathsf{decide}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi : \neg\Psi)) \qquad \wr \text{since } \mathcal{T} \text{ is decidable and deductive} \wr \quad \square
\end{aligned}
$$

But in many tools, decision of formulæ with universal quantifiers is problematic. If we want an exact resolution of satisfiability using just existential quantifiers, we need stronger hypotheses. One sufficient condition is that the theory is *complete*. In the context of classical first order logic, a theory can be defined to be complete if for all sentences Ψ in the language of the theory, either Ψ is in the theory or $\neg\Psi$ is in the theory.

$$
\mathsf{satisfiable}_\mathcal{T}(\Psi) \Leftrightarrow (\mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_\Psi : \Psi)) \qquad \text{(when } \mathcal{T} \text{ is decidable and complete)}
$$

Proof. Assume $\mathcal{T}$ is complete. Then, either $\exists \vec{\mathbf{x}}_\Psi : \Psi \in \mathcal{T}$, in which case $\mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_\Psi : \Psi)$ returns *true* and we conclude $\mathsf{satisfiable}_\mathcal{T}(\Psi)$. Or $\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T}$ so $\mathsf{decide}_\mathcal{T}(\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi))$ returns *true*. But if a sentence is in the theory, that means that for all models of that theory, the sentence is true, so:

$$
\begin{aligned}
& \neg\mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_\Psi : \Psi) \\
\Leftrightarrow\quad & \mathsf{decide}_\mathcal{T}(\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi)) \qquad\qquad \wr \mathcal{T} \text{ complete} \wr \\
\Leftrightarrow\quad & \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T} \qquad\qquad \wr \text{def. decision procedure} \wr \\
\Rightarrow\quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \\
& \qquad\qquad \wr \text{def. } \mathfrak{M}(\mathcal{T}') \triangleq \{I \in \mathfrak{I} \mid \forall \Psi' \in \mathcal{T}' : \exists \eta' : I \models_{\eta'} \Psi'\} \wr \\
\Rightarrow\quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \qquad \wr \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \text{ has no free variable} \wr
\end{aligned}
$$

$$\Leftrightarrow \quad \forall I \in \mathfrak{M}(\mathcal{T}) : \neg(\exists \eta : I \models_\eta \exists \vec{\mathbf{x}}_\Psi : \Psi) \qquad \wr\text{def. } \neg\wr$$

$$\Leftrightarrow \quad \forall I \in \mathfrak{M}(\mathcal{T}) : \neg(\exists \eta : I \models_\eta \Psi) \qquad \wr\text{def. } I \models_\eta \exists \mathbf{x} : \Psi \text{ in Sect. } \mathbf{2.3}\wr$$

$$\Leftrightarrow \quad \neg(\exists\, I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi) \qquad \wr\text{def. } \neg\wr$$

$$\Leftrightarrow \quad \neg\mathsf{satisfiable}_\mathcal{T}(\Psi) \qquad \wr\text{def. } \mathsf{satisfiable}_\mathcal{T}(\Psi) \triangleq \exists\, I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi\wr \quad \square$$

It might be the case that we only need the decision procedure to be equivalent to satisfiability for a subset of the language of the theory. Then the same proof can be applied. Partial completeness can be defined in the following way: a theory is *partially complete* for a set of formulæ A iff for all $\Psi \in A$, either Ψ is in the theory or $\neg\Psi$ is in the theory.

Decision procedures will be most useful to provide approximations of implication. But in general, one needs to know if an implication is valid, and most decision procedures can only decide existential sentences. Here is the way to use decision procedures to approximate the validity of implication:

$$\begin{aligned}
& \mathsf{valid}_\mathcal{T}(\forall \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \Rightarrow \Psi') \\
\triangleq \quad & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \forall \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \Rightarrow \Psi' && \wr\text{def. validity modulo } \mathcal{T}\wr \\
\Leftrightarrow \quad & \neg(\exists\, I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \exists \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \wedge \neg\Psi') && \wr\text{def. negation}\wr \\
\Leftrightarrow \quad & \neg(\mathsf{satisfiable}_\mathcal{T}(\Psi \wedge \neg\Psi')) && \wr\text{def. satisfiability modulo } \mathcal{T}\wr \\
\Rightarrow \quad & \neg\mathsf{decide}_\mathcal{T}(\exists \vec{\mathbf{x}}_{\Psi\wedge\neg\Psi'} : \Psi \wedge \neg\Psi') && \wr\text{when } \mathcal{T} \text{ is decidable and satisfiable.} \\
& && \text{Equivalence } \Leftrightarrow \text{ holds when } \mathcal{T} \text{ is complete for } \exists \vec{\mathbf{x}}_{\Psi\wedge\neg\Psi'} : \Psi \wedge \neg\Psi'\wr
\end{aligned}$$

2.7 Comparison of theories

Except for decision procedures, theories are equivalent when they have the same models. A theory $\mathcal{T}_1$ is *more general* than a theory $\mathcal{T}_2$ when all models of $\mathcal{T}_2$ are models of $\mathcal{T}_1$ i.e. $\mathfrak{M}(\mathcal{T}_2) \subseteq \mathfrak{M}(\mathcal{T}_1)$. A sufficient condition for $\mathcal{T}_1$ to be more general than $\mathcal{T}_2$ is $\mathcal{T}_1 \subseteq \mathcal{T}_2$ (since $\mathcal{T}_1 \subseteq \mathcal{T}_2$ implies $\mathfrak{M}(\mathcal{T}_2) \subseteq \mathfrak{M}(\mathcal{T}_1)$). The converse holds for deductive theories. The most general theory for a given signature is the theory of $\{\mathtt{tt}\}$ (or equivalently its deductive closure), also called the theory of logical validities. If a theory $\mathcal{T}_1$ is more general than $\mathcal{T}_2$, then we have, for all formula Ψ:

$$\mathsf{satisfiable}_{\mathcal{T}_2}(\Psi) \Rightarrow \mathsf{satisfiable}_{\mathcal{T}_1}(\Psi), \quad \text{and} \quad \mathsf{valid}_{\mathcal{T}_1}(\Psi) \Rightarrow \mathsf{valid}_{\mathcal{T}_2}(\Psi)$$

A consequence is that a decision procedure for a theory can be used to approximate the satisfiability in a more general theory. Another consequence is that the implication is less often true with a more general theory.

3 Terminology on Abstract Interpretation

3.1 Interpreted concrete semantics

Abstractions in abstract interpretation [18,20], are relative to an interpreted concrete semantics $C_\mathfrak{I}[\![\mathtt{P}]\!]$ of programs $\mathtt{P}$ as defined by a program interpretation $\mathfrak{I} \in \mathfrak{J}$. That concrete semantics is defined as the set of invariants of the programs, that is post-fixpoints

in a complete lattice/cpo of concrete properties $\langle \mathcal{P}_\mathfrak{I}, \subseteq \rangle$ and a concrete transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$. We define $\mathbf{postfp}^{\leq} f \triangleq \{ x \mid f(x) \leq x \}$.

$$\begin{array}{rl}
\mathcal{R}_\mathfrak{I} & \text{concrete observables}[9] \\
\mathcal{P}_\mathfrak{I} \triangleq \wp(\mathcal{R}_\mathfrak{I}) & \text{concrete properties} \\
F_\mathfrak{I}[\![\mathtt{P}]\!] \in \mathcal{P}_\mathfrak{I} \to \mathcal{P}_\mathfrak{I} & \text{concrete transformer of program P} \\
C_\mathfrak{I}[\![\mathtt{P}]\!] \triangleq \mathbf{postfp}^{\subseteq} F_\mathfrak{I}[\![\mathtt{P}]\!] \in \wp(\mathcal{P}_\mathfrak{I}) & \text{concrete semantics of program P}
\end{array}$$

where the concrete transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$ of program P is built out of the set primitives $\emptyset$, $\mathcal{R}_\mathfrak{I}$, $\cup$, $\cap$, ... and the forward and backward transformers $\mathsf{f}, \mathsf{b} \in \mathcal{P}_\mathfrak{I} \to \mathcal{P}_\mathfrak{I}$ for assignment, the transformer $\mathsf{p} \in \mathcal{P}_\mathfrak{I} \to \mathcal{B}$ for tests, ...

Note that if the concrete transformer admits a least fixpoint, then it is enough to consider only that least fixpoint and we don't need to compute the whole set of post-fixpoints (see also Sect. **3.4**).

Example 1. In the context of invariance properties for imperative languages with program interpretation $\mathfrak{I} \in \mathfrak{J}$, we can take a concrete state to be a function from variables [10] to elements in the set $\mathfrak{I}_\mathcal{V}$, so that properties are sets of such functions.

$$\begin{array}{ll}
\mathcal{R}_\mathfrak{I} \triangleq \mathbb{X} \to \mathfrak{I}_\mathcal{V} & \text{concrete environments} \\
\mathcal{P}_\mathfrak{I} \triangleq \wp(\mathcal{R}_\mathfrak{I}) & \text{concrete invariance properties}
\end{array}$$

The transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$ for the invariance semantics is defined by structural induction on the program P in terms of the complete lattice operations $\langle \wp(\mathcal{R}_\mathfrak{I}), \subseteq, \emptyset, \mathcal{R}_\mathfrak{I}, \cup, \cap \rangle$ and the following local invariance transformers

$$\begin{array}{rll}
\mathsf{f}_\mathfrak{I}[\![\mathtt{x} := e]\!]P \triangleq \{\eta[\mathtt{x} \leftarrow [\![e]\!]_\mathfrak{I}\eta] \mid \eta \in P)\} & & \text{Floyd's assignment post-condition} \\
\mathsf{b}_\mathfrak{I}[\![\mathtt{x} := e]\!]P \triangleq \{\eta \mid \eta[\mathtt{x} \leftarrow [\![e]\!]_\mathfrak{I}\eta] \in P\} & & \text{Hoare's assignment pre-condition} \\
\mathsf{p}_\mathfrak{I}[\![\varphi]\!]P \triangleq \{\eta \in P \mid [\![\varphi]\!]_\mathfrak{I}\eta = \mathit{true}\} & & \text{test} \qquad \square
\end{array}$$

Example 2. The program `P ≜ x=1; while true {x=incr(x)}` with the arithmetic interpretation $\mathfrak{I}$ on integers $\mathfrak{I}_\mathcal{V} = \mathbb{Z}$ has loop invariant $\mathbf{lfp}^{\subseteq} F_\mathfrak{I}[\![\mathtt{P}]\!]$ where $F_\mathfrak{I}[\![\mathtt{P}]\!](X) \triangleq \{\eta \in \mathcal{R}_\mathfrak{I} \mid \eta(\mathtt{x}) = 1\} \cup \{\eta[\mathtt{x} \leftarrow \eta(\mathtt{x}) + 1] \mid \eta \in X\}$. The increasing chain of iterates $F_\mathfrak{I}[\![\mathtt{P}]\!]^n = \{\eta \in \mathcal{R}_\mathfrak{I} \mid 0 < \eta(\mathtt{x}) < n\}$ has limit $\mathbf{lfp}^{\subseteq} F_\mathfrak{I}[\![\mathtt{P}]\!] = \bigcup_{n \geqslant 0} F_\mathfrak{I}[\![\mathtt{P}]\!]^n = \{\eta \in \mathcal{R}_\mathfrak{I} \mid 0 < \eta(\mathtt{x})\}$. $\square$

3.2 Abstract domains

In static analysis by abstract interpretation [18,20], abstract domains are used to encapsulate abstract program properties and abstract operations (including the logical lattice structure, elementary transformers, convergence acceleration operators, etc.). An abstract domain is therefore $\langle A, \sqsubseteq, \bot, \top, \sqcup, \sqcap, \nabla, \Delta, \bar{\mathsf{f}}, \bar{\mathsf{b}}, \bar{\mathsf{p}}, \ldots \rangle$ where

[9] Examples of observables are set of states, set of partial or complete execution traces, etc.
[10] maybe including the program counter etc.

$\overline{P}, \overline{Q}, \ldots \in A$	abstract properties
$\sqsubseteq \in A \times A \to \mathcal{B}$	abstract partial order [11]
$\bot, \top \in A$	infimum, supremum
$\sqcup, \sqcap, \nabla, \Delta \in A \times A \to A$	abstract join, meet, widening, narrowing
$\ldots$	
$\bar{\mathsf{f}} \in (\mathbb{x} \times \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \to A \to A$	abstract forward assignment transformer
$\bar{\mathsf{b}} \in (\mathbb{x} \times \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \to A \to A$	abstract backward assignment transformer
$\bar{\mathsf{p}} \in \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to A \to A$	abstract condition transformer

3.3 Abstract semantics

The abstract semantics of a program $\mathtt{P}$ is assumed to be given as a set of post-fixpoints $\overline{C}[\![\mathtt{P}]\!] \triangleq \{\overline{P} \mid \overline{F}[\![\mathtt{P}]\!](\overline{P}) \sqsubseteq \overline{P}\}$ or in least fixpoint form $\overline{C}[\![\mathtt{P}]\!] \triangleq \{\mathbf{lfp}^{\sqsubseteq}\, \overline{F}[\![\mathtt{P}]\!]\}$ (or, by the singleton isomorphism, the more frequent $\mathbf{lfp}^{\sqsubseteq}\, \overline{F}[\![\mathtt{P}]\!]$) when such a least fixpoint does exist (e.g. [46]) where $\overline{F}[\![\mathtt{P}]\!] \in A \to A$ is the abstract transformer of program $\mathtt{P}$ built out of the primitives $\bot, \top, \sqcup, \sqcap, \nabla, \Delta, \bar{\mathsf{f}}, \bar{\mathsf{b}}, \bar{\mathsf{p}}, \ldots$[12]. As was the case for the concrete semantics, we preferably use least fixpoints when that is possible.

3.4 Soundness of abstract domains

Soundness relates abstract properties to concrete properties using a function γ such that

$$\gamma \in A \xrightarrow{\nearrow} \mathcal{P}_{\mathfrak{I}} \quad \text{concretization [13]}$$

The soundness of abstract domains, is defined as, for all $\overline{P}, \overline{Q} \in A$,

$(\overline{P} \sqsubseteq \overline{Q}) \Rightarrow (\gamma(\overline{P}) \subseteq \gamma(\overline{Q}))$	order	$\gamma(\bot) = \emptyset$	infimum
$\gamma(\overline{P} \sqcup \overline{Q}) \supseteq (\gamma(\overline{P}) \cup \gamma(\overline{Q}))$	join	$\gamma(\top) = \top_{\mathfrak{I}}$	supremum [14]
...			

Observe that defining an abstraction consists in choosing the domain A of abstract properties and the concretization γ. So, this essentially consists in choosing a set of concrete properties $\gamma[A]$ (where $\gamma[X] \triangleq \{\gamma(x) \mid x \in X\}$) which can be exactly represented in the abstract while the other concrete properties $P \in \mathcal{P}_{\mathfrak{I}} \setminus \gamma[A]$ cannot and so must be over-approximated by some $\overline{P} \in A$ such that $P \subseteq \gamma(\overline{P})$. By assuming the existence of an element $\top$ of A with concretization $\top_{\mathfrak{I}}$, there always exists such a $\overline{P}$. For precision, the minimum one, or else the minimal ones, if any, are to be preferred.

---

[11] If $\sqsubseteq$ is a pre-order then A is assumed to be quotiented by the equivalence relation $\equiv \triangleq \sqsubseteq \cap \sqsubseteq^{-1}$.

[12] In general, this is more complex, with formulæ involving many fixpoints, but this simple setting already exhibits all difficulties.

[13] Given posets $\langle L, \sqsubseteq \rangle$ and $\langle P, \leqslant \rangle$, we let $L \xrightarrow{\nearrow} P$ to be the set of increasing (monotone, isotone, ...) maps of L into P.

[14] For example $\top_{\mathfrak{I}} \triangleq \mathcal{R}_{\mathfrak{I}}$ in the context of invariance properties for imperative languages.

3.5 Soundness of abstract semantics

The abstract semantics $\overline{C}[\![\mathtt{P}]\!] \in A$ is *sound* which respect to a concrete semantics $C[\![\mathtt{P}]\!]$ of a program P whenever

$$\forall \overline{P} \in A : (\exists \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \ \overline{C} \sqsubseteq \overline{P}) \Rightarrow (\exists C \in C[\![\mathtt{P}]\!] : \ C \subseteq \gamma(\overline{P}))$$

It is *complete* whenever $\forall \overline{P} \in A : (\exists C \in C[\![\mathtt{P}]\!] : C \subseteq \gamma(\overline{P})) \Rightarrow (\exists \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \overline{C} \sqsubseteq \overline{P})$. When the concrete and abstract semantics are defined in post-fixpoint form $C[\![\mathtt{P}]\!] \triangleq \mathbf{postfp}^{\subseteq} F[\![\mathtt{P}]\!]$, the soundness of the abstract semantics follows from the soundness of the abstraction in Sect. **3.4** and the soundness of the abstract transformer [18,20]

$$\forall \overline{P} \in A : F[\![\mathtt{P}]\!] \circ \gamma(\overline{P}) \subseteq \gamma \circ \overline{F}[\![\mathtt{P}]\!](\overline{P})$$

Example 3. Continuing *Ex. 1* in the context of invariance properties for imperative languages, the soundness of the abstract transformer generally follows from the following soundness conditions on local abstract transformers, for all $\overline{P} \in A$,

$$\begin{array}{rll} \gamma(\bar{\mathsf{f}}[\![\mathtt{x} := e]\!]\overline{P}) \supseteq \mathsf{f}_{\mathfrak{I}}[\![\mathtt{x} := e]\!]\gamma(\overline{P}) & & \text{assignment post-condition} \\ \gamma(\bar{\mathsf{b}}[\![\mathtt{x} := e]\!]\overline{P}) \supseteq \mathsf{b}_{\mathfrak{I}}[\![\mathtt{x} := e]\!]\gamma(\overline{P}) & & \text{assignment pre-condition} \\ \gamma(\bar{\mathsf{p}}[\![\varphi]\!]\overline{P}) \supseteq \mathsf{p}_{\mathfrak{I}}[\![\varphi]\!]\gamma(\overline{P}) & & \text{test} \end{array} \quad \square$$

Observe that soundness is preserved by composition of increasing concretizations.

4 Abstraction of Multi-Interpreted Concrete Semantics

The interpreted concrete semantics of Sect. **3.1** is relative to one interpretation $\mathfrak{I}$ of the programming language data, functions, and predicates. But the theories used in SMT solvers can have many different models, corresponding to possible interpretations. In fact, the same holds for programs: they can be executed on different platforms, and it can be useful to collect all the possible behaviors, e.g. to provide a more general proof of correctness.

4.1 Multi-interpreted semantics

In a *multi-interpreted semantics*, we will give semantics to a program P in the context of a set of interpretations $\mathcal{I}$. Then a program property in $\mathcal{P}_{\mathcal{I}}$ provides for each interpretation in $\mathcal{I}$, a set of program observables satisfying that property in that interpretation.

$$\begin{array}{ll} \mathcal{R}_I & \text{program observables} \\ \mathcal{P}_{\mathcal{I}} \triangleq I \in \mathcal{I} \mathrel{\not\mapsto} \wp(\mathcal{R}_I) & \text{interpreted properties} \\ \quad \simeq \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\})\ ^{15} & \end{array}$$

The multi-interpreted semantics of a program P in the context of $\mathcal{I}$ is

[15] A partial function $f \in A \rightarrow\!\!\!\!\!\rightarrow B$ with domain $\mathsf{dom}(f) \in \wp(A)$ is understood as the relation $\{\langle x, f(x)\rangle \in A \times B \mid x \in \mathsf{dom}(f)\}$ and maps $x \in A$ to $f(x) \in B$, written $x \in A \mathrel{\not\mapsto} f(x) \in B$ or $x \in A \mathrel{\not\mapsto} B_x$ when $\forall x \in A : f(x) \in B_s \subseteq B$.

$$C_{\mathcal{I}}[\![\mathtt{P}]\!] \in \wp(\mathcal{P}_{\mathcal{I}})$$

In the context of invariance properties for imperative languages with multiple program interpretations $\mathcal{I} \in \wp(\mathfrak{I})$, *Ex. 1* can be generalized by taking

$$\mathcal{R}_I \triangleq \mathbb{x} \to I_{\mathcal{V}} \quad \text{concrete interpreted environments}$$

The transformer $F_{\mathcal{I}}[\![\mathtt{P}]\!]$ for the invariance semantics is defined by structural induction on the program P in terms of the complete lattice operations $\langle \mathcal{P}_{\mathcal{I}}, \subseteq, \emptyset, \top_{\mathcal{I}}, \cup, \cap \rangle$ where $\top_{\mathcal{I}} \triangleq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\}$ and the following local invariance transformers

$$\begin{array}{rll}
\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \triangleq \{\langle I, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \mid I \in \mathcal{I} \wedge \langle I, \eta\rangle \in P)\} & & \text{assignment post-condition} \\
\mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \triangleq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \langle I, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \in P\} & & \text{assignment pre-condition} \\
\mathsf{p}_{\mathcal{I}}[\![\varphi]\!]P \triangleq \{\langle I, \eta\rangle \in P \mid I \in \mathcal{I} \wedge [\![\varphi]\!]_I \eta = \mathit{true}\} & & \text{test}
\end{array}$$

In particular for $\mathcal{I} = \{\mathfrak{I}\}$, we get the transformers of *Ex. 1*, up to the isomorphism $\iota_{\mathfrak{I}}(P) \triangleq \{\langle \mathfrak{I}, \eta\rangle \mid \eta \in P\}$ with inverse $\iota_{\mathfrak{I}}^{-1}(Q) \triangleq \{\eta \mid \langle \mathfrak{I}, \eta\rangle \in Q\}$.

The natural ordering to express abstraction (or precision) on multi-interpreted semantics is the subset ordering, which gives a lattice structure to the set of multi-interpreted properties: a property P_1 is more abstract than P_2 when $P_2 \subset P_1$, meaning that P_1 allows more behaviors for some interpretations, and maybe that it allows new interpretations. Following that ordering, we can express systematic abstractions of the multi-interpreted semantics.

4.2 Abstractions between multi-interpretations

If we can only compute properties on one interpretation $\mathfrak{I}$, as in the case of Sect. **3.1**, then we can approximate a multi-interpreted program saying that we know the possible behaviors when the interpretation is $\mathfrak{I}$ and we know nothing (so all properties are possible) for the other interpretations of the program. On the other hand, if we analyze a program that can only have one possible interpretation with a multi-interpreted property, then we are doing an abstraction in the sense that we add more behaviors and forget the actual property that should be associated with the program. So, in general, we have two sets of interpretations, one $\mathcal{I}$ is the context of interpretations for the program and the other $\mathcal{I}^\sharp$ is the set of interpretations used in the analysis. The relations between the two is a Galois connection $\langle \mathcal{P}_{\mathcal{I}}, \subseteq\rangle \underset{\alpha_{\mathcal{I}\to\mathcal{I}^\sharp}}{\overset{\gamma_{\mathcal{I}^\sharp\to\mathcal{I}}}{\leftrightarrows}} \langle \mathcal{P}_{\mathcal{I}^\sharp}, \subseteq\rangle$ where

$$\begin{array}{l}
\alpha_{\mathcal{I}\to\mathcal{I}^\sharp}(P) \triangleq P \cap \mathcal{P}_{\mathcal{I}^\sharp} \\
\gamma_{\mathcal{I}^\sharp\to\mathcal{I}}(Q) \triangleq \left\{ \langle I, \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I, \eta\rangle \in Q\right) \right\}
\end{array}$$

Proof. Suppose $P \in \mathcal{P}_{\mathcal{I}}$ and $Q \in \mathcal{P}_{\mathcal{I}^\sharp}$. Then

$$\begin{array}{lll}
& \alpha_{\mathcal{I}\to\mathcal{I}^\sharp}(P) \subseteq Q & \\
\Leftrightarrow & P \cap \mathcal{P}_{\mathcal{I}^\sharp} \subseteq Q & \wr\text{def. } \alpha_{\mathcal{I}\to\mathcal{I}^\sharp}\int \\
\Leftrightarrow & \forall \langle I, \eta\rangle \in P \cap \mathcal{P}_{\mathcal{I}^\sharp} : \langle I, \eta\rangle \in Q & \wr\text{def. } \subseteq\int
\end{array}$$

$$\Leftrightarrow \quad \forall \langle I,\, \eta\rangle \in P : \langle I,\, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I,\, \eta\rangle \in Q \qquad \wr \text{def. } \cap \int$$

$$\Leftrightarrow \quad \forall \langle I,\, \eta\rangle \in P,\, I \in \mathcal{I} \wedge (\langle I,\, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I,\, \eta\rangle \in Q) \qquad \wr P \in \mathcal{P}_{\mathcal{I}} \int$$

$$\Leftrightarrow \quad P \subseteq \{\langle I,\, \eta\rangle \mid I \in \mathcal{I} \wedge (\langle I,\, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I,\, \eta\rangle \in Q)\} \qquad \wr \text{def. } \subseteq \int$$

$$\Leftrightarrow \quad P \subseteq \gamma_{\mathcal{I}^\sharp \to \mathcal{I}}(Q) \qquad \wr \text{def. } \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \int \quad \square$$

Note that if the intersection of $\mathcal{I}^\sharp$ and $\mathcal{I}$ is empty then the abstraction is trivially $\emptyset$ for all properties, and if $\mathcal{I} \subseteq \mathcal{I}^\sharp$ then the abstraction is the identity.

Considering the soundness of transformers defined in Sect. **3.5** for the forward assignment of Sect. **4.1**, we get, for all $P^\sharp \in \mathcal{P}_{\mathcal{I}^\sharp}$,

$$\begin{array}{ll}
& \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!] \circ \gamma_{\mathcal{I}^\sharp \to \mathcal{I}}(P^\sharp) \\
= & \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \;\middle|\; \langle I,\, \eta\rangle \in \gamma_{\mathcal{I}^\sharp \to \mathcal{I}}(P^\sharp) \right\} \\
& \qquad \wr \text{def. } \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \triangleq \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \;\middle|\; \langle I,\, \eta\rangle \in P) \right\} \int \\
= & \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \;\middle|\; I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta\rangle \in P^\sharp\right) \right\} \qquad \wr \text{def. } \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \int \\
= & \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \;\middle|\; I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow (I \in \mathcal{I}^\sharp \wedge \langle I,\, \eta\rangle \in P^\sharp)\right) \right\} \qquad \wr \text{def. } \Rightarrow \int \\
\subseteq & \left\{ \langle I,\, \eta'\rangle \;\middle|\; I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta'\rangle \in \left\{ \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \;\middle|\; I \in \mathcal{I}^\sharp \wedge \langle I,\, \eta\rangle \in P^\sharp \right\}\right) \right\} \\
& \qquad \wr \text{def. } \subseteq \int \\
= & \left\{ \langle I,\, \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta\rangle \in \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp)\right) \right\} \\
& \qquad \wr \text{by defining } \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!] \in \mathcal{P}_{\mathcal{I}^\sharp} \stackrel{\nearrow}{\to} \mathcal{P}_{\mathcal{I}^\sharp} \text{ such that} \\
& \qquad\qquad \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!]P^\sharp \triangleq \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]\rangle \;\middle|\; I \in \mathcal{I}^\sharp \wedge \langle I,\, \eta\rangle \in P^\sharp \right\} \\
& \qquad \int \\
\subseteq & \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \circ \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp) \qquad \wr \text{def. } \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \quad . \int
\end{array}$$

Observe that $\mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!]$ and $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]$ have exactly the same definition. However, the corresponding post-fixpoint semantics do differ when $\mathcal{I}^\sharp \neq \mathcal{I}$ since $\langle \mathcal{P}_{\mathcal{I}^\sharp}, \subseteq\rangle \neq \langle \mathcal{P}_{\mathcal{I}}, \subseteq\rangle$.

4.3 Uniform abstraction of interpretations

In some cases, we describe the properties of the program without distinguishing the interpretations in the context of the program. This is the case for example when expressing properties that should hold for all interpretations which are possible for the program. That abstraction simply forgets the interpretations and just keeps the union of all the possible behaviors.

Example 4. That is what the Astrée analyzer [23] does when taking all possible rounding error modes for floating points computations. $\square$

The abstraction is described by $\langle \mathcal{P}_{\mathcal{I}},\, \subseteq\rangle \underset{\alpha_{\mathcal{I}}}{\overset{\gamma_{\mathcal{I}}}{\leftrightarrows}} \langle \cup_{I \in \mathcal{I}} \mathcal{R}_I,\, \subseteq\rangle$ where

$$\begin{array}{l}
\gamma_{\mathcal{I}}(E) \triangleq \left\{ \langle I,\, \eta\rangle \;\middle|\; \eta \in E \right\} \\
\alpha_{\mathcal{I}}(P) \triangleq \left\{ \eta \;\middle|\; \exists I :\, \langle I,\, \eta\rangle \in P \right\}
\end{array}$$

4.4 Abstraction by a theory

Another direction for abstraction is to keep the context of interpretations and forget about the properties on variables. This is simply a projection on the first component of the pairs of interpretation and environment. In some cases it can be difficult to represent exactly an infinite set of interpretations, and we can use theories (preferably deductive with a recursively enumerable number of axioms) to represent the set of interpretations which are models of that theories. The relationship between theories and multi-interpreted semantics is expressed by the concretization function:

$$\gamma_{\mathfrak{M}}(\mathcal{T}) \triangleq \left\{ \langle I,\ \eta \rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \right\}$$

Notice, though, that because the lattice of theories is not complete, there is no best abstraction of a set of interpretations by a theory in general.

Example 5. If $\mathfrak{I}$ interprets programs over the natural numbers, then by Gödel's first incompleteness theorem there is no enumerable first-order theory characterizing this interpretation, so the poset has no best abstraction of $\{\mathfrak{I}\}$. □

Once an (arbitrary) theory $\mathcal{T}$ has been chosen to abstract a set $\mathcal{I}$ of interpretations there is a best abstraction $\alpha_{\mathcal{I}\to\gamma_{\mathfrak{M}}(\mathcal{T})}(P)$ of interpreted properties in $P \in \mathcal{P}_{\mathcal{I}}$ by abstract properties in $\mathcal{P}_{\gamma_{\mathfrak{M}}(\mathcal{T})}$. However there might be no finite formula to encode this best abstraction.

5 Uninterpreted Axiomatic Semantics

We consider Hoare's axiomatic semantics [13] of a simple imperative language. The language is completely uninterpreted so programs are merely program schemata with no constraints imposed on the interpretation of functions $\mathbb{f}$ and predicates $\mathbb{p}$ [37]. Program properties are specified by formulæ in $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ which is a lattice $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}), \Rrightarrow \rangle$ for the pre-order $(\Psi \Rrightarrow \Psi') \triangleq \mathsf{valid}(\Psi \Rightarrow \Psi')$ hence is considered to be quotiented by $(\Psi \Lleftarrow\Rrightarrow \Psi') \triangleq \mathsf{valid}(\Psi \Leftrightarrow \Psi')$. The axiomatic semantics $C^{\mathfrak{a}}[\![\mathtt{P}]\!]$ of a program $\mathtt{P}$ is assumed to be defined in post-fixpoint form [16] $C^{\mathfrak{a}}[\![\mathtt{P}]\!] \triangleq \left\{ \Psi \;\middle|\; F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \Rrightarrow \Psi \right\}$ where $F_{\mathfrak{a}}[\![\mathtt{P}]\!] \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \xrightarrow{\nearrow} \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ is the predicate transformer of program $\mathtt{P}$ such that $F_{\mathfrak{a}}[\![\mathtt{P}]\!](I) \Rrightarrow I$ is the verification condition for $I \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ to be an inductive invariant for program $\mathtt{P}$. The program transformer $F_{\mathfrak{a}}$ maps programs $\mathtt{P}$ to a predicate transformer $F_{\mathfrak{a}}[\![\mathtt{P}]\!]$ which we assume to be defined in terms of primitive operations $\mathtt{false}, \mathtt{true}, \vee, \wedge, \mathsf{f}_{\mathfrak{a}}, \mathsf{b}_{\mathfrak{a}}, \mathsf{p}_{\mathfrak{a}}, \ldots$ such that

$\mathsf{f}_{\mathfrak{a}} \in (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ $\mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x']$	axiomatic forward assignment transformer
$\mathsf{b}_{\mathfrak{a}} \in (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ $\mathsf{b}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \Psi[\mathtt{x} \leftarrow t]$	axiomatic backward assignment transformer
$\mathsf{p}_{\mathfrak{a}} \in \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathcal{B}$ $\mathsf{p}_{\mathfrak{a}}[\![\varphi]\!]\Psi \triangleq \Psi \wedge \varphi$	axiomatic transformer for program test of condition φ

Example 6. Continuing *Ex. 2*, consider the signature with equality and one unary function, $\mathbb{f} \triangleq \mathbb{f}^0 \cup \mathbb{f}^1$ with $\mathbb{f}^0 \triangleq \{\mathtt{0}\}$, $\mathbb{f}^1 \triangleq \{\mathtt{incr}\}$ and $\mathbb{p} \triangleq \emptyset$[16]. By Ehrenfeucht's theorem [26], this first order logic is decidable.

The program $\mathtt{P} \triangleq \mathtt{x{=}0;\ while\ true\ \{x{=}incr(x)\}}$ has loop invariant $\mathbf{lfp}^{\Mapsto} F_{\mathfrak{a}}[\![\mathtt{P}]\!]$ where $F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \triangleq (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr(x)}[\mathtt{x} \leftarrow x']) \Leftrightarrow (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$. The fixpoint iterates are

— $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^0 \triangleq \mathtt{false}$ ⟅basis⟆

— $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^1 \triangleq F_{\mathfrak{a}}[\![\mathtt{P}]\!](F_{\mathfrak{a}}[\![\mathtt{P}]\!]^0)$

$= (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \mathtt{false}[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$

⟅def. $F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \triangleq (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$⟆

$\Leftrightarrow (\mathtt{x} = \mathtt{0})$

— $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^2 \triangleq F_{\mathfrak{a}}[\![\mathtt{P}]\!](F_{\mathfrak{a}}[\![\mathtt{P}]\!]^1)$

$= (\mathtt{x} = \mathtt{0}) \vee (\exists x_2 : (x_2 = \mathtt{0}) \wedge \mathtt{x} = \mathtt{incr}(x_2))$ ⟅def. $F_{\mathfrak{a}}[\![\mathtt{P}]\!]$⟆

$\Leftrightarrow (\mathtt{x} = \mathtt{0}) \vee (\mathtt{x} = \mathtt{incr(0)})$ ⟅simplification⟆

— $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n \triangleq \bigvee_{i=0}^{n-1} (\mathtt{x} = \mathtt{incr}^i\mathtt{(0)})$ ⟅induction hypothesis⟆

⟅where the abbreviations are $\bigvee \emptyset \triangleq \mathtt{false}$, $\mathtt{incr}^0\mathtt{(0)} \triangleq \mathtt{0}$, $\mathtt{incr}^{n+1}\mathtt{(0)} \triangleq \mathtt{incr(incr}^n\mathtt{(0))}$, and $\bigvee_{i=1}^{k} \varphi_i \triangleq \varphi_1 \vee \cdots \vee \varphi_k$ so that $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$⟆

— $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^{n+1} \triangleq F_{\mathfrak{a}}[\![\mathtt{P}]\!](F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n)$ ⟅recurrence⟆

$= (\mathtt{x} = \mathtt{0}) \vee (\exists x' : F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$ ⟅def. $F_{\mathfrak{a}}[\![\mathtt{P}]\!]$⟆

$= (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \big(\bigvee_{i=0}^{n-1} (\mathtt{x} = \mathtt{incr}^i\mathtt{(0)})\big)[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = \mathtt{incr}(x'))$ ⟅by ind. hyp.⟆

$= (\mathtt{x} = \mathtt{0}) \vee (\exists x' : \big(\bigvee_{i=0}^{n-1} (x' = \mathtt{incr}^i\mathtt{(0)})\big) \wedge \mathtt{x} = \mathtt{incr}(x'))$ ⟅def. substitution⟆

$\Leftrightarrow \bigvee_{i=0}^{n} (\mathtt{x} = \mathtt{incr}^i\mathtt{(0)}) \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ ⟅simplification and def. $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$⟆

All iterates $F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n$, $n \geqslant 0$ of $F_{\mathfrak{a}}[\![\mathtt{P}]\!]$ belong to $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ and form an increasing chain for $\Mapsto$ in the poset $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}), \Mapsto \rangle$ up to equivalence $\Longleftrightarrow$. Notice above that $\bigvee_{i=1}^{n} \Psi_i$ is not a formula of $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ but a notation, only used in the mathematical reasoning, to denote a finite formula of $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$, precisely $\Psi_1 \vee \ldots \vee \Psi_n$. In the axiomatic semantics, the least fixpoint does not exist and the set of post-fixpoints is $\left\{ F_{\mathfrak{a}}[\![\mathtt{P}]\!]^n(\mathtt{true}) \;\middle|\; n \geq 0 \right\}$ $= \{\mathtt{true},\ \mathtt{x} = 0 \vee (\exists x' : \mathtt{x} = \mathtt{incr}(x'),\ \ldots\}$.

Of course the intuition would be that $\mathbf{lfp}^{\Mapsto} F_{\mathfrak{a}}[\![\mathtt{P}]\!] = \bigvee_{i \geqslant 0} (\mathtt{x} = \mathtt{incr}^i\mathtt{(0)})$, which is an infinite formula, hence not in $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$. There is therefore a fundamental incompleteness in using $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ to express invariant of loops in programs on $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$.

[16] A possible interpretation would be $\mathfrak{I}_{\mathcal{V}} \triangleq \mathbb{N}$, $\gamma(\mathtt{0}) = 0$ and $\gamma(\mathtt{incr})(x) = x + 1$, another one would be with ordinals, integers, non-standard integers, or even lists where $\mathtt{0}$ is `null` and `incr(x)` returns a pointer to a node with a single field pointing to `x`.

On one hand there may be *infinitely* many exact iterates all expressible in $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ and on the other hand their *limit* cannot be expressed in $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ [10,13]. Because of this problem, refinement with predicate abstraction would typically never terminate (unless a widening is used [1]). □

6 Axiomatic Semantics Modulo Interpretations

The axiomatic semantics of Sect. **5** is completely uninterpreted that is a purely syntactic object. A link between the uninterpreted axiomatic semantics of Sect. **5** and the interpreted concrete semantics used in Sect. **3.1** or even better Sect. **4.1** must be established [10,13]. We do this in two steps, first giving universal interpretation and then considering semantics modulo a restricted set of interpretations for more precise results.

6.1 Universal interpretation of the axiomatic semantics

The universal interpretation consists in describing the properties encoded by a formula on all possible interpretations. Thus, the concretization of a formula will be given by:

$$\gamma^{\mathfrak{a}} \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \xrightarrow{\nearrow} \mathcal{P}_{\mathfrak{Z}}$$
$$\gamma^{\mathfrak{a}}(\Psi) \triangleq \{\langle I,\, \eta\rangle \mid I \models_\eta \Psi\}$$

By definition of $I \models_\eta \Psi$, $\gamma^{\mathfrak{a}}$ is increasing in that for all $\Psi, \Psi' \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$, $\Psi \Rrightarrow \Psi'$ implies that $\gamma^{\mathfrak{a}}(\Psi) \subseteq \gamma^{\mathfrak{a}}(\Psi')$.

Theorem 1. *The axiomatic semantics is sound with respect to all multi-interpreted semantics.*

Proof. To prove the soundness of the axiomatic semantics with respect to the multi-interpreted semantics in a context of interpretations $\mathcal{I}$, we first prove the soundness in the context of all possible interpretations and then use the result of Sect. **4.2** to show soundness with respect to all contexts of interpretations, which include the basic case of Sect. **3.1**.

Soundness with respect to a multi-interpreted semantics of the program can be expressed as:

$$\forall \Psi \in C^{\mathfrak{a}}[\![\mathtt{P}]\!] :\ C_{\mathfrak{Z}}[\![\mathtt{P}]\!] \subseteq \gamma^{\mathfrak{a}}(\Psi)$$

This can be proven by verifying local soundness conditions. For the forward assignment,

$$\begin{aligned}
& \gamma^{\mathfrak{a}}(\mathtt{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi) \\
\triangleq\ & \gamma^{\mathfrak{a}}(\exists x' :\ \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x']) \\
& \qquad\qquad \wr \text{def. } \mathtt{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \exists x' :\ \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x'] \int \\
=\ & \{\langle I,\, \eta\rangle \mid I \models_\eta (\exists x' :\ \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x'])\}\ \wr \text{def. } \gamma^{\mathfrak{a}}(\Psi) = \{\langle I,\, \eta\rangle \mid I \models_\eta \Psi\} \int \\
=\ & \{\langle I,\, \eta'[\mathtt{x} \leftarrow [\![t]\!]_{\prime}\eta']\rangle \mid I \models_{\eta'} \Psi\}
\end{aligned}$$

$\wr$since $I \models_\eta (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x'])$ if and only if $\exists \eta' : I \models_{\eta'} \Psi$ and $\eta = \eta'[\mathtt{x} \leftarrow [\![t]\!]_I \eta']\wr$

$= \quad \{\langle I, \eta[\mathtt{x} \leftarrow [\![t]\!]_I \eta]\rangle \mid \langle I, \eta\rangle \in \{\langle I, \eta\rangle \mid I \models_\eta \Psi\}\} \qquad \wr$renaming η' into η and def. $\in\wr$

$= \quad \{\langle I, \eta[\mathtt{x} \leftarrow [\![t]\!]_I \eta]\rangle \mid \langle I, \eta\rangle \in \gamma^{\mathfrak{a}}(\Psi)\} \qquad \wr$def. $\gamma^{\mathfrak{a}}(\Psi) \backsimeq \{\langle I, \eta\rangle \mid I \models_\eta \Psi\}\wr$

$= \quad \mathsf{f}_{\mathfrak{Z}}[\![\mathtt{x} := t]\!] \circ \gamma^{\mathfrak{a}}(\Psi) \qquad \wr$def. $\mathsf{f}_{\mathfrak{Z}}[\![\mathtt{x} := t]\!]P \triangleq \{\langle I, \eta[\mathtt{x} \leftarrow [\![t]\!]_I \eta]\rangle \mid \langle I, \eta\rangle \in P\}\wr$

and similarly for backward assignment and test. □

6.2 Adding more precision: axiomatic semantics modulo interpretations

An abstraction which is sound for all possible interpretations of a program is very likely to be imprecise on the interpretations we are interested in. An easy way to be more precise is to accept as invariants all post-fixpoints for $\Rightarrow_{\mathcal{I}}$ instead of just $\Rightarrow$. That defines the axiomatic semantics modulo interpretations.

Definition 1. An axiomatic semantics $C^{\mathfrak{a}}_{\mathcal{I}}[\![\mathrm{P}]\!]$ modulo interpretations $\mathcal{I}$ of a program P is defined as the uninterpreted axiomatic semantics of Sect. **5** with respect to the lattice $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}),\ \Rightarrow_{\mathcal{I}}\rangle$ for the pre-order $(\Psi \Rightarrow_{\mathcal{I}} \Psi') \triangleq \mathsf{valid}_{\mathcal{I}}(\Psi \Rightarrow \Psi')$ (or equivalently $\neg\mathsf{satisfiable}_{\mathcal{I}}(\Psi \wedge \neg\Psi')$) understood as quotiented by $(\Psi \Leftrightarrows_{\mathcal{I}} \Psi') \triangleq \mathsf{valid}_{\mathcal{I}}(\Psi \Leftrightarrow \Psi')$. □

A remarkable point of these axiomatic semantics modulo interpretations $\mathcal{I}$ is that they all share the same program transformer $F_{\mathfrak{a}}[\![\mathrm{P}]\!]$, but indeed, if $\mathcal{I} \subseteq \mathcal{I}'$, then $C^{\mathfrak{a}}_{\mathcal{I}}[\![\mathrm{P}]\!] \supseteq C^{\mathfrak{a}}_{\mathcal{I}'}[\![\mathrm{P}]\!] \cap \mathcal{P}_{\mathcal{I}}$ meaning that semantics modulo $\mathcal{I}$ is more precise than modulo $\mathcal{I}'$.

Soundness of the axiomatics semantics modulo interpretations $\mathcal{I}$ is a consequence of the fact that it is the reduced product of the axiomatic semantics with the abstraction of the interpretations of the program. Such abstraction is trivially sound as soon as $\mathcal{I}$ contains all the interpretations in the semantics of the program we wish to consider.

6.3 Axiomatic Semantics Modulo Theory

An axiomatic semantics $C^{\mathfrak{a}}_{\mathcal{I}}[\![\mathrm{P}]\!]$ modulo interpretations $\mathcal{I}$ of Def. **1** may be definable by a theory $\mathcal{T}$ which models define $\mathcal{I}$.

Definition 2. An axiomatic semantics $C^{\mathfrak{a}}_{\mathcal{T}}[\![\mathrm{P}]\!]$ of a program P modulo a theory $\mathcal{T}$ is defined as $C^{\mathfrak{a}}_{\mathcal{T}}[\![\mathrm{P}]\!] = C^{\mathfrak{a}}_{\mathfrak{M}(\mathcal{T})}[\![\mathrm{P}]\!]$ [17]. □

Soundness again follows from the soundness of the abstraction by theory (Sect. **4.4**). There is no such best abstraction, but any sound abstraction will give sound abstraction for the axiomatic semantics modulo theory.

Since bigger sets of interpretations mean less precise semantics, more general theories will give less precise invariants.

[17] that is, by Def. **1**, as the uninterpreted axiomatic semantics of Sect. **5** with respect to the lattice $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}),\ \Rightarrow_{\mathcal{T}}\rangle$ for the pre-order $(\Psi \Rightarrow_{\mathcal{T}} \Psi') \triangleq \mathsf{valid}_{\mathcal{T}}(\Psi \Rightarrow \Psi')$ (or equivalently $\neg\mathsf{satisfiable}_{\mathcal{T}}(\Psi \wedge \neg\Psi')$) understood as quotiented by $(\Psi \Leftrightarrows_{\mathcal{T}} \Psi') \triangleq \mathsf{valid}_{\mathcal{T}}(\Psi \Leftrightarrow \Psi')$.

6.4 Interpreted assignment

A consequence of considering semantics modulo interpretations $\mathcal{I}$ is the ability to use simpler predicate transformers. In particular the axiomatic transformer for $\mathtt{x} := t$ is simpler if the interpretations of term t in $\mathcal{I}$ are all such that the t is invertible for variable $\mathtt{x}$. A term t is *invertible* for variable $\mathtt{x}$, with inverse $t_{\mathtt{x}}^{-1}$, in a set $\mathcal{I}$ of interpretations iff the formula $\forall \mathtt{y} : \forall \mathtt{z} : (\mathtt{y} = t[\mathtt{x} \leftarrow \mathtt{z}]) \Leftrightarrow (\mathtt{z} = t_{\mathtt{x}}^{-1}[\mathtt{x} \leftarrow \mathtt{y}])$ is true for all interpretations in $\mathcal{I}$. In this case the axiomatic assignment forward transformer can be simplified into

$$\mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \Psi[\mathtt{x} \leftarrow t_{\mathtt{x}}^{-1}] \qquad \text{assignment postcondition, } t \text{ invertible into } t_{\mathtt{x}}^{-1} \text{ under } \mathcal{I}$$

$$\begin{array}{lll}
 & \gamma_{\mathcal{I}}^{\mathfrak{a}}(\mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi) & t \text{ invertible into } t_{\mathtt{x}}^{-1} \text{ under } \mathcal{I} \\
\triangleq & \gamma_{\mathcal{I}}^{\mathfrak{a}}(\Psi[\mathtt{x} \leftarrow t_{\mathtt{x}}^{-1}]) & \wr \text{def. } \mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \Psi[\mathtt{x} \leftarrow t_{\mathtt{x}}^{-1}] \wr \\
= & \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge I \models_\eta \Psi[\mathtt{x} \leftarrow t_{\mathtt{x}}^{-1}]\} & \wr \text{def. } \gamma_{\mathcal{I}}^{\mathfrak{a}}(\Psi) \backsimeq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge I \models_\eta \Psi\} \wr \\
= & \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \exists x' : I \models_\eta \exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge x' = t_{\mathtt{x}}^{-1}[\mathtt{x} \leftarrow \mathtt{x}]\} & \wr \text{def. } \exists \wr \\
= & \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge I \models_\eta (\exists x' : \Psi[\mathtt{x} \leftarrow x']) \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x']\} & \\
 & & \wr t \text{ is invertible for } \mathtt{x}, \text{ with inverse } t_{\mathtt{x}}^{-1} \wr \\
= & \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := t]\!] \circ \gamma_{\mathcal{I}}^{\mathfrak{a}}(\Psi) &
\end{array}$$

Observe that the uninterpreted axiomatic semantics transformer $F_{\mathfrak{a}}[\![\mathtt{P}]\!]$ using the invertible assignment postcondition in any program $\mathtt{P}$ is no longer sound for all possible classes of interpretations $\mathcal{I} \in \wp(\mathfrak{I})$ but only for those for which inversion is possible.

7 Logical Abstract Domains

When performing program verification in the first-order logic setting, computing the predicate transformer is usually quite immediate. The two hard points are (1) the computation of the least fixpoint (or an approximation of it since the logical lattice is not complete) and (2) proving that the final formula implies the desired property. To solve the first case, a usual (but not entirely satisfactory) solution is to restrict the set of formulæ used to represent environments such that the ascending chain condition is enforced. For the proof of implication, a decidable theory is used. So we define logical abstract domains in the following general setting:

Definition 3. A *logical abstract domain* is a pair $\langle A, \mathcal{T}\rangle$ of a set $A \in \wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}))$ of logical formulæ and of a theory $\mathcal{T}$ of $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ (which is decidable (and deductive) (and complete on A)). The abstract properties $\Psi \in A$ define the concrete properties $\gamma_{\mathcal{T}}^{\mathfrak{a}}(\Psi) \triangleq \left\{ \langle I, \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta \Psi \right\}$ relative to the models $\mathfrak{M}(\mathcal{T})$ of theory $\mathcal{T}$. The abstract order $\sqsubseteq$ on the abstract domain $\langle A, \sqsubseteq\rangle$ is defined as $(\Psi \sqsubseteq \Psi') \triangleq ((\forall \vec{\mathtt{x}}_\Psi \cup \vec{\mathtt{x}}_{\Psi'} : \Psi \Rightarrow \Psi') \in \mathcal{T})$. □

This definition of logical abstract domains is close to the logical abstract interpretation framework developped by Gulwani and Tiwari [33,32]. The main difference with our approach is that we give semantics with respect to a concrete semantics corresponding to the actual behavior of the program, whereas in the work of Gulwani and Tiwari,

the behavior of the program is assumed to be described by formulæ in the same theory as the theory of the logical abstract domain. Our approach allows the description of the abstraction mechanism, comparisons of logical abstract domains, and to provide proofs of soundness on a formal basis.

7.1 Abstraction to Logical Abstract Domains

Because $A \in \wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}))$, we need to approximate formulæ in $\wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \setminus A$ by a formula in A. The alternatives [21] are either to choose a context-dependent abstraction (a different abstraction is chosen in different circumstances, which can be understood as a widening [12]) or to define an abstraction function to use a uniform context-independent approximation whenever needed. For example, the abstract assignment would be $\mathsf{f}^\sharp[\![\mathtt{x} := t]\!]\varphi \triangleq \mathsf{alpha}^{\mathcal{I}}_A(\mathsf{f}[\![\mathtt{x} := t]\!]\varphi)$. The abstraction

$$\mathsf{alpha}^{\mathcal{I}}_A \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to A \quad \text{abstraction (function/algorithm)}$$

abstracts a concrete first-order logic formula appearing in the axiomatic semantics into a formula in the logical abstract domain A. It is assumed to be sound in that

$$\forall \Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}),\ \forall I \in \mathcal{I} :\ I \models \Psi \Rightarrow \mathsf{alpha}^{\mathcal{I}}_A(\Psi) \quad \text{soundness} \tag{1}$$

It should also preferably be computable (in which case we speak of an abstraction algorithm, which can be used in the abstract semantics whereas when it is not computable it has to be eliminated e.g. by manual design of an over-approximation of the abstract operations).

Example 7 (Literal elimination). Assume that the axiomatic semantics is defined on $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ and that the logical abstract domain is $A = \mathbb{F}(\mathbb{x}, \mathbb{f}_A, \mathbb{p}_A)$ where $\mathbb{f}_A \subseteq \mathbb{f}$ and $\mathbb{p}_A \subseteq \mathbb{p}$. The abstraction $\mathsf{alpha}^{\mathcal{I}}_A(\Psi)$ of $\Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ can be obtained by repeating the following approximations until stabilization.

- If the formula Ψ contains one or several occurrences of a term $t \in \mathbb{f} \setminus \mathbb{f}_A$ (so is of the form $\Psi[t, \ldots, t]$), they can all be approximated by $\exists x : \Psi[x, \ldots, x]$;
- If the formula Ψ contains one or several occurrences of an atomic formula $a \in \mathbb{p} \setminus \mathbb{p}_A$ (so is of the form $\Psi[a, \ldots, a]$), this atomic formula can be replaced by `true` in the positive positions and by `false` in the negative positions.

In both cases, this implies soundness (1) and the algorithm terminates since Ψ is finite. □

Example 8 (Quantifier elimination). If the abstract domain $A \subseteq \mathbb{C}(\mathbb{x}, \mathbb{f}_A, \mathbb{p}_A)$ is quantifier-free then the quantifiers must be eliminated which is possible without loss of precision in some theories such as Presburger arithmetic (but with a potential blow-up of the formula size see e.g. [11,28,27]). Otherwise, besides simple simplifications of formulæ (e.g. replacing $\exists x : x = t \wedge \Psi[x]$ by $\Psi[t]$), a very coarse abstraction to $A \subseteq \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ would eliminate quantifiers bottom up, putting the formula in disjunctive normal form and eliminating the literals containing existentially quantified variables (or dually [39]),

again with a potential blow-up. Other proposals of abstraction functions (often not identified as such) include the quantifier elimination heuristics defined in Simplify [25, Sect. 5], [24, Sect. 6], or the (doubly-exponential) methods of [30,31] (which might even be made more efficient when exploiting the fact that an implication rather than an equivalence is required). □

Example 9 (Interval abstraction). Let us consider the mimimal abstraction α_m (which reduced product with the maximal abstraction, yields the interval abstraction [17,18]).

$$\begin{aligned}\alpha_m(\Psi) &\triangleq \bigwedge_{\mathtt{x}\in\mathbb{x}} \{\mathtt{c} \leqslant \mathtt{x} \mid \mathtt{c} \in \mathbb{c} \wedge \mathit{min}(\mathtt{c}, \mathtt{x}, \Psi)\}\\ \mathit{min}(\mathtt{c}, \mathtt{x}, \Psi) &\triangleq \forall \mathtt{x} : (\exists \vec{\mathtt{x}}_\Psi \setminus \{\mathtt{x}\} : \Psi) \Rightarrow (\mathtt{c} \leqslant \mathtt{x}) \wedge\\ &\qquad \forall \mathtt{m} : (\forall \mathtt{x} : (\exists \vec{\mathtt{x}}_\Psi \setminus \{\mathtt{x}\} : \Psi) \Rightarrow (\mathtt{m} \leqslant \mathtt{x})) \Rightarrow \mathtt{m} \leqslant \mathtt{c}\end{aligned}$$

Replacing the unknown constant $\mathtt{c}$ by a variable c in $\mathit{min}(\mathtt{c}, \mathtt{x}, \Psi)$, a solver might be able to determine a suitable value for $\mathtt{c}$. Otherwise the maximality requirement of $\mathtt{c}$ might be dropped to get a coarser abstraction and `true` returned in case of complete failure of the solver. □

7.2 Logical abstract transformers

For soundness with respect to a set of interpretations $\mathcal{I}$, the abstract transformers must be chosen such that [18,20]

$\mathsf{f}^\sharp \in (\mathbb{x} \times \mathbb{T}) \to A \to A$	abstract forward assignment transformer
$\forall \varphi \in A, \forall I \in \mathcal{I} :: I \models \mathsf{f}[\![\mathtt{x} := t]\!]\varphi \Rightarrow \mathsf{f}^\sharp[\![\mathtt{x} := t]\!]\varphi$	abstract postcondition
$\mathsf{b} \in (\mathbb{x} \times \mathbb{T}) \to A \to A$	abstract backward assignment transformer
$\forall \varphi \in A, \forall I \in \mathcal{I} :: I \models \mathsf{b}[\![\mathtt{x} := t]\!]\varphi \Rightarrow \mathsf{b}^\sharp[\![\mathtt{x} := t]\!]\varphi$	abstract precondition
$\mathsf{p} \in \mathbb{L} \to A \to A$	condition abstract transformer
$\forall \varphi \in A, \forall I \in \mathcal{I} : \mathsf{p}[\![I]\!]\varphi \Rightarrow \mathsf{p}^\sharp[\![I]\!]\varphi$	abstract test

It follows from the definition of the uninterpreted axiomatic semantics in Sect. **5** that we can define the abstract transformer to be the axiomatic transformer (under suitable closure hypothesis on A in which case they map a formula in A to a formula in A), otherwise using the abstraction α of Sect. **7.1** or using a specific local abstraction. In addition, such abstraction may be necessary on joins, as A may contain just formulæ without $\vee$.

Example 10 (Transfer function abstraction). In many static analyzers, the abstract transfer functions $\mathsf{f}^\sharp$, $\mathsf{b}^\sharp$, and $\mathsf{p}^\sharp$ of Sect. **3.2** are most often designed, proved correct and implemented by hand. This design and implementation would better be totally automatized, going back to the manual solution when automation is too inefficient or imprecise.

For logical abstract domains, the best transformers are $\mathsf{f}^\sharp[\![\mathtt{x} := t]\!] \triangleq \alpha \circ \mathsf{f}[\![\mathtt{x} := t]\!]$, $\mathsf{b}^\sharp[\![\mathtt{x} := t]\!] \triangleq \alpha \circ \mathsf{b}[\![\mathtt{x} := t]\!]$, and $\mathsf{p}^\sharp[\![\varphi]\!] \triangleq \alpha \circ \mathsf{p}[\![\varphi]\!]$ using the uninterpreted axiomatic transformers of Sect. **5**. Abstract transformers or some over-approximations (e.g. $\alpha \circ$

$\mathsf{f}[\![\mathtt{x} := t]\!] \dot{\Rightarrow} \mathsf{f}^\sharp[\![\mathtt{x} := t]\!]$) might be automatically computable using solvers (see e.g. [43] when A satisfies the ascending chain condition).

Continuing *Ex. 9*, where the abstract domain is $A = \{\bigwedge_{\mathtt{x}\in\mathbb{x}} \mathtt{c_x} \leqslant \mathtt{x} \mid \forall \mathtt{x} \in \mathbb{x} : \mathtt{c_x} \in \mathbb{f}^0\}$ and $\alpha = \alpha_m$, a SMT solver (e.g. with linear arithmetics or even simple inequalities [42]) might be usable when restricting Ψ in $\alpha_m(\Psi)$ to the formulæ obtained by the transformation of formulæ of A by the uninterpreted axiomatic transformers of Sect. **5**. □

Example 11 (Abstract assignment). The non-invertible assignment transformer returns a quantified formula

$$\mathsf{f}[\![\mathtt{x} := t]\!]\Psi \triangleq \exists x' : \Psi[\mathtt{x}/x'] \wedge \mathtt{x} = t[\mathtt{x}/x'] \quad \text{non-invertible assignment}$$

which may have to be abstracted to A can be done using the abstraction α of Sect. **7.1** or the widening of Sect. **7.3** or on the fly, using program specificities. For example, in straight line code outside of iteration or recursion, the existential quantifier can be eliminated

- using logical equivalence, by Skolemization where $\forall x_1 : \ldots \forall x_n : \exists y : p(x_1, \ldots, x_n, y)$ is replaced by the equi-satisfiable formula $\forall x_1 : \ldots \forall x_n : p(x_1, \ldots, x_n, f_y(x_1, \ldots, x_n))$ where f_y is a fresh symbol function;
- using a program transformation, since x' denotes the value of the variable $\mathtt{x}$ before the assignment we can use a program equivalence introducing new fresh program variable $\mathtt{x}'$ to store this value since "$\mathtt{x} := t$" is equivalent to "$\mathtt{x}' := \mathtt{x};\ \mathtt{x} := t[\mathtt{x} \leftarrow \mathtt{x}']$" [18]. We get

  $$\mathsf{f}^\sharp[\![\mathtt{x} := t]\!]\varphi \triangleq \varphi[\mathtt{x} \leftarrow \mathtt{x}'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow \mathtt{x}'] \quad \text{abstract non-invertible assignment}$$

  which may be a formula in A. This ensures soundness by program equivalence.

These local solutions cannot be used with iterations or recursions (but with a k-limiting abstraction as in bounded model checking) since a fresh auxiliary function/variable is needed for each iteration/recursive call, which number may be unbounded. □

7.3 Widening and narrowing

When the abstract domain does not satisfy the ascending chain condition, a widening is needed both to cope with the absence of infinite disjunctions and to enforce the convergence of fixpoint interation [18,12]. Designing a universal widening for logical abstract domain is difficult since powerful widenings prevent infinite evolutions in the semantic computation, evolutions which are not always well reflected as a syntactic evolution in logical abstract domains. Nevertheless, we can propose several possible widenings .

1. Widen to a finite sub-domain W of A organized in a partial order choosing $X \triangledown Y$ to be $\Psi \in W$ such that $Y \Rightarrow \Psi$ and starting from the smallest elements of W (or use a further abstraction into W as in Sect. **7.1**);

[18] This is similar to but different from Skolemization since we use auxiliary program variables instead of auxiliary functions.

2. Limit the size of formulæ to $k > 0$, eliminating new literals in the simple conjunctive normal form appearing beyond the fixed maximal size (e.g. depth) k (the above widenings are always correct but not very satisfactory, see [22]);
3. Follow the syntactic evolution of successive formulæ and reduce the evolving parts as proposed by [38] for Typed Decision Graphs.
4. Make generalizations (e.g. $l(1) \vee l(2) \vee \ldots$ implies $\exists k \geqslant 0 : l(k)$ and abstract the existential quantifier, see *Ex. 8*) or use saturation [19] [14].

8 Soundness of Unsound Abstractions

As noted in Sect. **6**, the axiomatic semantics modulo theory $\mathcal{T}$ (and thus the logical abstract domains with that semantics) are sound when all the interpretations we wish to consider for the program are models of $\mathcal{T}$. But what we see in practice is that the actual interpretations corresponding to the machine execution of programs are not models of the theories used in the program proofs. Typical examples include proofs on natural numbers, whereas the size of integers are bounded, or reasoning on floating point arithmetics as if floats behaved as reals. Indeed, it already happened that the Astrée analyzer found a buffer overrun in programs formally "proven" correct, but with respect to a theory that was an unsound approximation of the program semantics.

Still, such reasonings can give some informations about the program provided the invariant they find is precise enough. One way for them to be correct for an interpretation $\mathfrak{I}$ is to have one model I of the theory to agree with $\mathfrak{I}$ on the formulæ that appear during the computation. Formally, two theories I_1 and I_2 agree on Ψ when $\left\{\eta \,\middle|\, I_1 \models_\eta \Psi\right\} = \left\{\eta \,\middle|\, I_2 \models_\eta \Psi\right\}$

This can be achieved by monitoring the formulæ during the computation, for example insuring that the formulæ imply that numbers are always smaller than the largest machine integer. It is enough to perform this monitoring during the invariant checking phase ($F_\mathfrak{a}[\![\mathrm{P}]\!](\Psi) \rightmodels_{\mathcal{T}} \Psi$), so we can just check for Ψ and $F_\mathfrak{a}[\![\mathrm{P}]\!](\Psi)$, but in some case, it can be worthwhile to detect early that the analysis cannot be correct because of an initial difference between one of the concrete interpretations and the models of the theory used to reason about the program.

9 Conclusion

The idea of using a universal representation of program properties by first-order formulæ originates from mathematical logics and underlies all deductive methods since [29,35,40] and its first automation [36]. Similarly, BDDs [5] are the universal representation used in symbolic model-checking [8].

In contrast, the success and difficulty of abstract interpretation relies on leaving open the computer representation of program properties by reasoning only in terms of abstract domains that is their algebraic structures. Data representations and algorithms have to be designed to implement the abstract domains, which is flexible and allows

[19] Saturation means to compute the closure of a given set of formulas under a given set of inference rules.

for specific efficient implementations. The alternative of using a universal representation and abstraction of the program collecting semantics for static analysis can also be considered [15]. It was intensively and successfully exploited e.g. in TVLA [45] or by abstract compilation to functional languages [3] or Prolog [9] but with difficulties to scale up.

One advantage of logical abstract domains with a uniform representation of program properties as first-order logic formulæ is the handy and understandable interface to interact with the end-user (at least when the formulæ are small enough to remain readable). It is quite easy for the end-user to help the analysis e.g. by providing assertions, invariants, hypotheses, etc. In contrast, the end-user has no access to the complex internal representation of program properties in algebraic abstract domains and so has no way to express internal properties used during the analysis, except when the abstract domain explicitly provides such an interface, e.g. for trace partitioning [44], quaternions [2], etc.

One disadvantage of the uniform representation of program properties in logical abstract domains is that it can be inefficient in particular because the growth of formulæ may be difficult to control (and may require a widening). In particular, first order-logic is the source of incompleteness at the level of the collecting semantics. The relative completeness result [10] assumes expressiveness, that is, the algebraic fixpoint semantics can be expressed in the first-order logic, which is rarely the case. Using an incomplete basis for abstraction means that some problems cannot be solved by any abstraction. Indeed, we have shown that logical abstract domains are an abstraction of the multi-interpreted semantics, and to prove more properties on that semantics, we could use second order logics [34], which is then complete but not decidable, or an ad-hoc collection of algebraic abstract domains.

The best choice, though, would be a combination of both algebraic and logical abstract domains, so that we get the best of both worlds. A possible way to investigate in that direction could be the combination of the Nelson-Oppen procedure for combining theories [41] on one hand and the reduced product on the other hand [20].

References

1. T. Ball, A. Podelski, and S.K. Rajamani. Relative completeness of abstraction refinement for software model checking. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, pages 158—172, Grenoble, April 8–12, 2002. LNCS 2280, Springer, Heidelberg.
2. J. Bertrane, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010, Atlanta, Georgia*, pages AIAA 2010–3385. AIAA, 20–22 April 2010.
3. D. Boucher and M. Feeley. Abstract compilation: a new implementation paradigm for static analysis. In T. Gyimothy, editor, *Proc. 6th Int. Conf. on Compiler Construction, CC '96*, Linköping, Lecture Notes in Computer Science 1060, pages 192–207. Springer, Heidelberg, April 24–26, 1996.
4. A.R. Bradley and Z. Manna. *The Calculus of Computation, Decision procedures with Applications to Verification*. Springer, Heidelberg, 2007.

5. Randal E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.
6. C.C. Chang and H.J. Keisler. Model theory. volume 73 of *Studies in logic and the foundation of mathematics*, New York, 1990. Elsevier Science.
7. E.M. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.
8. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, Cambridge, 1999.
9. M. Codish and H. Søndergaard. Meta-circular abstract interpretation in Prolog. In T. Mogensen, D. Schmidt, and I.H. Sudburough, editors, *The Essence of Computation: Complexity, Analysis, Transformation*, volume 2566 of *LNCS*, pages 109–134. Springer-Verlag, 2002.
10. S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7:70–80, 1978.
11. D.C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 91(7):91–99, 1972.
12. P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, 21 March, 1978.
13. P. Cousot. Methods and logics for proving programs. In J. van Leeuwen, editor, *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, chapter 15, pages 843–993. Elsevier Science Publishers B.V., Amsterdam, 1990.
14. P. Cousot. Verification by abstract interpretation. In N. Dershowitz, editor, *Proc. 23rd Int. Col., ICALP '96*, pages 1–3. LNCS 1099, Springer, Heidelberg, Paderborn, Germany, July 8–12, 1996.
15. P. Cousot. Abstract interpretation based static analysis parameterized by semantics. In P. Van Hentenryck, editor, *Pro. 4th Int. Symp. on Static Analysis, SAS '97*, Paris, LNCS 1302, pages 388–394. Springer, Heidelberg, September 8–10, 1997.
16. 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.
17. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. 2nd Int. Symp. on Programming*, pages 106–130, Paris, 1976. Dunod, Paris.
18. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, 1977. ACM Press.
19. P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.
20. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, 1979. ACM Press.
21. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.
22. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *Proc. 4th Int. Symp. Programming Language Implementation and Logic Programming, PLILP '92*, Leuven, 26–28 August 1992, LNCS 631, pages 269–295. Springer, Heidelberg, 1992.
23. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astrée analyser. In M. Sagiv, editor, *Proc. 14th European Symp. on Programming Languages and Systems, ESOP '2005, Edinburg*, LNCS 3444, pages 21–30. Springer, Heidelberg, April 2–10, 2005.
24. L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In A. Voronkov, editor, *Proc. 15th Computer-Aided Verification conf. (CAV'03)*, volume 2725 of *LNCS*, pages 14–26. Springer, Heidelberg, 2003.

25. D. Detlefs, G. Nelson, and J.B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM (JACM)*, 52(3):365–473, 2005.
26. A. Ehrenfeucht. Decidability of the theory of one function. *Notices Amer. Math. Soc.*, 6:268, 1959.
27. J. Ferrante and J.R. Geiser. An efficient decision procedure for the theory of rational order. *Theoretical Computer Science*, 4(2):227–233, 1977.
28. J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal of Computation*, 4(1):69–76, 1975.
29. R.W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Pro. Symp. in Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, Providence, 1967.
30. Y. Ge, C. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Conf. on Automated Deduction, CADE 21*, 4603 of LNAI:167–182, 2007.
31. Y. Ge and L. de Moura. Complete instantiation of quantified formulas in satisfiability modulo theories. *Computer Aided Verification, CAV'2009*, 5643 of LNCS:306–320, 2009.
32. S. Gulwani, B. McCloskey, and A. Tiwari. Lifting abstract interpreters to quantified logical domains. In *35th POPL*, pages 235–246, San Francisco, 2008. ACM Press.
33. S. Gulwani and A. Tiwari. Combining abstract interpreters. In M.I. Schwartzbach and T. Ball, editors, *PLDI 2006*, pages 376–386, Ottawa, Ontario, Canada, 11–14 June 2006. ACM Press.
34. P. Hitchcock and D. Park. Induction rules and termination proofs. In M. Nivat, editor, *Proc. 1st Int. Colloq. on Automata, Languages and Programming*, pages 225–251. North-Holland, 1973.
35. C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12(10):576–580, oct 1969.
36. J.C. King. *A Program Verifier*. PhD thesis, Carnegie-Mellon University, 1969.
37. D.C. Luckham, D.M.R. Park, and M.S. Paterson. On formalized computer programs. *J. of Computer and System Science*, 4(3):220–240, June 1970.
38. L. Mauborgne. Abstract interpretation using typed decision graphs. *Science of Computer Programming*, 31(1):91–112, May 1998.
39. K.L. McMillan. Applying SAT methods in unbounded symbolic model checking. In E. Brinksma and K.G. Larsen, editors, *Computer Aided Verification, CAV'2002*, volume 2404 of LNCS, pages 250–264, 2002.
40. P. Naur. Proofs of algorithms by general snapshots. *BIT*, 6:310–316, 1966.
41. G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, oct 1979.
42. V.R. Pratt. Two easy theories whose combination is hard. Technical report, MIT, september 1, 1977. `boole.stanford.edu/pub/sefnp.pdf`.
43. T.W. Reps, S. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In B. Steffen and G. Levi, editors, *Proc. 5th Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI 2004)*, pages 252–266, Venice, January 11–13, 2004. LNCS 2937, Springer, Heidelberg.
44. X. Rival and L. Mauborgne. The trace partitioning abstract domain. *ACM Transactions on Programming Languages and Systems*, 29(5), August 2007.
45. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. In *26th POPL*, pages 238–252, an Antonio, 1999. ACM Press.
46. A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.