A Framework for Combining Algebraic and Logical Abstract Interpretations

Patrick Cousot, Radhia Cousot, Laurent Mauborgne

A Framework for Combining Algebraic and Logical Abstract Interpretations

Patrick Cousot
ENS, INRIA & NYU
cousot@ens.fr, pcousot@cs.nyu.edu

Radhia Cousot
CNRS, ENS & MSR Redmond
radhia.cousot@ens.fr

Laurent Mauborgne
IMDEA
laurent.mauborgne@imdea.org

July 16, 2010

Abstract

We introduce a reduced product combining algebraic and logical abstractions to design program correctness verifiers and static analyzers by abstract interpretation.

1 Introduction

Recent progress in SMT solvers and theorem provers as used in program verification (where the inductive argument necessary for the proof is either provided by the end-user or by refinement of the specification) [2] has been recently exploited for static analysis by abstract interpretation [6, 7] (where the inductive argument necessary for the proof is computed directly (e.g. by elimination) or iteratively with convergence acceleration by widening/narrowing) using logical abstract domains [17]. Because of efficiency restrictions of SMT solvers and theorem provers, the analyzers hardly scale up beyond small programs. Moreover, because of effectiveness restrictions of SMT solvers and theorem provers, the program semantics which is used for the soundness proof is a mathematical semantics which significantly differs from the implementation semantics of the programming language. For example the theory of integer arithmetics is considered instead of modular arithmetics on 32 or 64 bits, or the theories of reals or rationals are considered instead of floats. It follows that the static analysis is unsound

for the machine semantics. Of course, modular arithmetics could be encoded with integer arithmetics and floats with rationals, but then SMT solvers and theorem provers would become highly inefficient. Static analyzers such as Astrée [1, 10] which are based on algebraic abstractions of the machine semantics do not have such efficiency and soundness limitations. It is therefore interesting not only to use SMT solvers and theorem provers in logical abstract domains but also to combine algebraic and logical abstract interpretations to get the best of both worlds i.e. scalability, expressivity, natural interface with the end-user using logical formulæ, and soundness with respect to the machine semantics. The proposed combination is based on the reduced product [7] which is commonly used in algebraic abstract interpreters (e.g. in Astrée [11]) while logical abstract interpreters combine (disjoint, convex, stably-infinite) theories by Nelson-Oppen procedure [23]. The key new idea is to show that Nelson-Oppen procedure computes a reduced product in an observational semantics, so that algebraic and logical abstract interpretations can naturally be combined in a classical way using a reduced product on this observational semantics. The main practical benefit is that reductions can be performed within the logical abstract domains, within the algebraic abstract domains, and also between the logical and the algebraic abstract domains, including the case of abstractions evolving during the analysis.

2 Syntax and semantics of programs

2.1 Sorts

We assume that we are given a set $\mathbb{s}$ of sorts $s \in \mathbb{s}$ corresponding to the notion of types in programming languages (e.g. $\mathbb{s} \triangleq \{\texttt{bool}, \texttt{char}, \texttt{int}, \ldots\}$) [31]. The unsorted case would have only one sort.

2.2 Signatures

A *sorted signature* is a tuple $\Sigma = \langle \mathbb{s}, \mathbb{x}, \mathbb{f}, \mathbb{p}, \# \rangle$ such that the sets $\mathbb{s}$ of sorts, $\mathbb{x}$ of variables, $\mathbb{f}$ of function symbols, and $\mathbb{p}$ of predicate symbols are mutually disjoints, the symbols have fixed arities $\mathbb{f} = \bigcup_{n \geqslant 0} \mathbb{f}^n$, $\mathbb{p} \triangleq \bigcup_{n \geqslant 0} \mathbb{p}^n$, and sorts defined by # such that

$\texttt{x}, \texttt{y}, \texttt{z}, \ldots \in \mathbb{x}$	variables $\texttt{x}$ of sort $\#(\texttt{x}) \in \mathbb{s}$
$\texttt{a}, \texttt{b}, \texttt{c}, \ldots \in \mathbb{f}^0$	constants $\texttt{c}$ of sort $\#(\texttt{c}) \in \mathbb{s}$
$\texttt{f}, \texttt{g}, \texttt{h}, \ldots \in \mathbb{f}^n$	function symbols $\texttt{f}$ of arity $n \geqslant 1$ and sort $\#(\texttt{f}) \in \mathbb{s}^n \to \mathbb{s}$
$\texttt{p}, \texttt{q}, \texttt{r}, \ldots \in \mathbb{p}^n$	predicate symbols $\texttt{p}$ of arity $n \geqslant 0$ and sort $\#(\texttt{p}) \in \mathbb{s}^n \to \{\texttt{bool}\}$.

As in first-order logics with equality, there is a distinguished predicate $= (t_1, t_2)$ for all sorts which we write $t_1 = t_2$. A *subsignature* is Σ' such that $\Sigma' \dot{\subseteq} \Sigma$, componentwise.

2.3 Syntax

The syntax of sorted first-order logic formulæ is as follows.

$t \in \mathbb{T}(\Sigma)$ terms

$t ::= \mathtt{x} \mid \mathtt{c} \mid \mathtt{f}(t_1, \ldots, t_n)$

$\#(t) = \#(\mathtt{x})$

$\#(t) = \#(\mathtt{c})$

$\#(t) = s$ where

$\#(\mathtt{f}) = s_1 \times \ldots \times s_n \to s$, $n \geqslant 1$ with $\#(t_i) = s_i$ for all $i = 1, \ldots, n$.

$a \in \mathbb{A}(\Sigma)$ atomic formulæ

$a ::= \mathtt{ff} \mid \mathtt{p}(t_1, \ldots, t_n) \mid \neg a$

$\#(a) = \mathtt{bool}$ where

$\#(\mathtt{p}) = s_1 \times \ldots \times s_n \to \mathtt{bool}$, $n \geqslant 1$ with $\#(t_i) = s_i$ for all $i = 1, \ldots, n$.

Clauses are quantifier-free formulæ in simple conjunctive normal form.

$\varphi, \psi, \ldots \in \mathbb{C}(\Sigma)$ clauses

$\varphi ::= a \mid \varphi \wedge \varphi$

$\#(\varphi) = \mathtt{bool}$.

First-order logic formulæ may be quantified.

$\Psi, \Phi, \ldots \in \mathbb{F}(\Sigma)$ quantified first-order formulæ

$\Psi ::= a \mid \neg\Psi \mid \Psi \wedge \Psi \mid \exists\, \mathtt{x} : \Psi$.

Finally programs of the programming language on a given signature Σ are built out of basic expressions and imperative commands.

$e, \ldots \in \mathbb{E}(\Sigma) \triangleq \mathbb{T}(\Sigma) \cup \mathbb{A}(\Sigma)$ program expressions

$C, \ldots \in \mathbb{L}(\Sigma)$ commands

$C ::= \mathtt{x} := e$ assignment, $\#(\mathtt{x}) = \#(e)$

$\mid \varphi$ test, $\#(\varphi) = \mathtt{bool}$.

Tests appear in conditionals and loops which syntax, as well as that of programs, are irrelevant. First-order logic formulæ may also appear in programs but only as specifications (e.g. for loop invariants).

2.4 Interpretations

An *interpretation* I for a signature $\Sigma = \langle \mathbb{s}, \mathbb{x}, \mathbb{f}, \mathbb{p}, \# \rangle$ is a pair $\langle I_{\mathcal{V}}, I_\gamma \rangle$ such that $I_{\mathcal{V}} = \bigcup_{s \in \mathbb{s}} I^s_{\mathcal{V}}$, $I^s_{\mathcal{V}}$ is a non-empty set of values of sort $s \in \mathbb{s}$, $\forall \mathtt{c} \in \mathbb{f}^0 : I_\gamma(\mathtt{c}) \in I^s_{\mathcal{V}}$ where $\#(\mathtt{c}) = s$, $\forall n \geqslant 1 : \forall \mathtt{f} \in \mathbb{f}^n : I_\gamma(\mathtt{f}) \in I^{s_1}_{\mathcal{V}} \times \ldots \times I^{s_n}_{\mathcal{V}} \to I^s_{\mathcal{V}}$ where $\#(\mathtt{f}) = s_1 \times \ldots \times s_n \to s$ and $\forall n \geqslant 0 : \forall \mathtt{p} \in \mathbb{p}^n : I_\gamma(\mathtt{p}) \in s_1 \times \ldots \times s_n \to \mathcal{B}$ [1] where $\#(\mathtt{p}) = s_1 \times \ldots \times s_n \to \mathtt{bool}$. Let $\mathfrak{I}(\Sigma)$ be the class of all such interpretations I. In a given interpretation $I \in \mathfrak{I}(\Sigma)$, an environment [2] is a function from variables to values with same sort

$$\eta \in \mathcal{R}^\Sigma_I \quad \triangleq \quad \mathbb{x} \to I_{\mathcal{V}} : \mathtt{x} \in \mathbb{x} \mapsto I^{\#(\mathtt{x})}_{\mathcal{V}} \qquad \text{sorted environments .}$$

An interpretation I and an environment $\eta \in \mathcal{R}^\Sigma_I$ satisfy a formula Ψ, written $I \models_\eta \Psi$, in the following way:

[1] $\mathcal{B} \triangleq \{false, true\}$ are the Boolean values, $s_1 \times \ldots \times s_n \triangleq \emptyset$, and $\emptyset \to B \simeq B$.

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

$$
\begin{array}{rcl@{\qquad}rcl}
I \models_\eta a & \triangleq & [\![a]\!]_I\eta & I \models_\eta \Psi \wedge \Psi' & \triangleq & (I \models_\eta \Psi) \wedge (I \models_\eta \Psi') \\
I \models_\eta \neg\Psi & \triangleq & \neg(I \models_\eta \Psi) & I \models_\eta \exists\, \mathtt{x} : \Psi & \triangleq & \exists\, v \in I_{\mathcal{V}} :\ I \models_{\eta[\mathtt{x}\leftarrow v]} \Psi \ [3]
\end{array}
$$

where the value $[\![a]\!]_I\eta \in \mathcal{B}$ of an atomic formula $a \in \mathbb{A}(\Sigma)$ in environment $\eta \in \mathcal{R}^\Sigma_I$ is

$$
\begin{array}{rcl}
[\![\mathtt{ff}]\!]_I\eta & \triangleq & \mathit{false} \\
[\![\mathtt{p}(t_1, \ldots, t_n)]\!]_I\eta & \triangleq & I_\gamma(\mathtt{p})([\![t_1]\!]_I\eta, \ldots, [\![t_n]\!]_I\eta), \quad n \geqslant 1 \\
[\![\neg a]\!]_I\eta & \triangleq & \neg[\![a]\!]_I\eta, \quad \text{where } \neg\mathit{true} = \mathit{false},\ \neg\mathit{false} = \mathit{true}
\end{array}
$$

and the value $[\![t]\!]_I\eta \in I^s_{\mathcal{V}}$ of the term $t \in \mathbb{T}(\Sigma)$ such that $\#(t) = s$ in environment $\eta \in \mathcal{R}^\Sigma_I$ is

$$
\begin{array}{rcl}
[\![\mathtt{x}]\!]_I\eta & \triangleq & \eta(\mathtt{x}) \\
[\![\mathtt{c}]\!]_I\eta & \triangleq & I_\gamma(\mathtt{c}) \\
[\![\mathtt{f}(t_1, \ldots, t_n)]\!]_I\eta & \triangleq & I_\gamma(\mathtt{f})([\![t_1]\!]_I\eta, \ldots, [\![t_n]\!]_I\eta)\ .
\end{array}
$$

In addition, in first-order logics 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 $I_{\mathcal{V}}$ encoded by its characteristic function.

2.5 Multi-interpreted program semantics

A *multi-interpreted semantics* assigns meanings to a program $\mathtt{P}$ in the context of a set of interpretations $\boldsymbol{\mathcal{I}} \in \wp(\mathfrak{I}(\Sigma))$ for the program signature $\Sigma = \langle \mathbb{s},\ \mathbb{x},\ \mathbb{f},\ \mathbb{p},\ \# \rangle$.

Example 1. Integers can have a mathematical interpretation or a modular interpretation on machines. □

Then a program property in $\mathfrak{P}^\Sigma_{\boldsymbol{\mathcal{I}}}$ provides for each interpretation in $\boldsymbol{\mathcal{I}}$, a set of environments for variables $\mathbb{x}$ satisfying that property in that interpretation.

$$
\begin{array}{rcll}
\mathfrak{R}^\Sigma_{\boldsymbol{\mathcal{I}}} & \triangleq & \left\{ \langle I,\ \eta \rangle \,\middle|\, I \in \boldsymbol{\mathcal{I}} \wedge \eta \in \mathcal{R}^\Sigma_I \right\} & \text{multi-interpreted environments} \\
\mathfrak{P}^\Sigma_{\boldsymbol{\mathcal{I}}} & \triangleq & \wp(\mathfrak{R}^\Sigma_{\boldsymbol{\mathcal{I}}}) & \text{multi-interpreted properties.}
\end{array}
$$

The multi-interpreted semantics of a program $\mathtt{P}$ in the context of multi-interpretations $\boldsymbol{\mathcal{I}}$ is assumed to be defined in fixpoint form

$$
\begin{array}{rcll}
C^\Sigma_{\boldsymbol{\mathcal{I}}}[\![\mathtt{P}]\!] & \in & \mathfrak{P}^\Sigma_{\boldsymbol{\mathcal{I}}} & \text{concrete semantics} \\
F^\Sigma_{\boldsymbol{\mathcal{I}}}[\![\mathtt{P}]\!] & \in & \mathfrak{P}^\Sigma_{\boldsymbol{\mathcal{I}}} \stackrel{\nearrow}{\to} \mathfrak{P}^\Sigma_{\boldsymbol{\mathcal{I}}} & \text{concrete transformer} \\
C^\Sigma_{\boldsymbol{\mathcal{I}}}[\![\mathtt{P}]\!] & \triangleq & \mathbf{lfp}^\subseteq F^\Sigma_{\boldsymbol{\mathcal{I}}}[\![\mathtt{P}]\!] & \text{least fixpoint semantics}
\end{array}
$$

[3] $\eta[\mathtt{x} \leftarrow v]$ is the assignment of v to $\mathtt{x}$ in η where $\#(\mathtt{x}) = \#(v)$, $\eta[\mathtt{x} \leftarrow v](\mathtt{x}) \triangleq v$, and $\eta[\mathtt{x} \leftarrow v](\mathtt{y}) \triangleq \eta(\mathtt{y})$ when $\mathtt{x} \neq \mathtt{y}$.

The transformer $F^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!]$ is assumed to be increasing. Since $\langle \mathfrak{P}^{\Sigma}_{\mathcal{I}}, \subseteq, \emptyset, \mathfrak{R}^{\Sigma}_{\mathcal{I}}, \cup, \cap \rangle$ is a complete lattice, $\mathbf{lfp}^{\subseteq} F^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!]$ does exist by [28]. The transformer $F^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!]$ is defined by structural induction on the program P in terms of the complete lattice operations and the following local transformers for the assignment postcondition

$$\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \quad \triangleq \quad \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \;\middle|\; I \in \mathcal{I} \wedge \langle I,\, \eta \rangle \in P \right\}$$

the assignment precondition

$$\mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \quad \triangleq \quad \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \in P \right\}$$

and tests

$$\mathsf{p}_{\mathcal{I}}[\![\varphi]\!]P \quad \triangleq \quad \left\{ \langle I,\, \eta \rangle \in P \;\middle|\; I \in \mathcal{I} \wedge [\![\varphi]\!]_I \eta = \mathit{true} \right\} .$$

Let $\mathfrak{I} \in \mathfrak{Z}(\Sigma)$ be the *program standard interpretation* e.g. as defined explicitly by a standard or implicitly by a compiler, linker, loader, operating system and network of machines. The standard concrete semantics is $\mathcal{I} = \{\mathfrak{I}\}$ in which case the local transformers correspond to Floyd/Hoare/Dijkstra predicate transformers, up to the isomorphism $\iota_{\mathfrak{I}}(P) \triangleq \{\langle \mathfrak{I},\, \eta \rangle \mid \eta \in P\}$ with inverse $\iota^{-1}_{\mathfrak{I}}(Q) \triangleq \{\eta \mid \langle \mathfrak{I},\, \eta \rangle \in Q\}$.

The reason why we consider multi-interpretations, and not just a single interpretation as it is classical, is that it is the natural setting for the logical abstract domains which are valid up to a theory (Sect. **2.11**), which can have many different interpretations.

2.6 Algebraic abstract domains

We let $\langle A^{\Sigma}_{\mathcal{I}}, \sqsubseteq, \top, \sqcup, \ldots, \bar{\mathsf{f}}, \bar{\mathsf{p}}, \ldots \rangle$ be an *abstract domain* abstracting multi-interpreted properties in $\mathfrak{P}^{\Sigma}_{\mathcal{I}}$ for signature Σ and multi-interpretations $\mathcal{I}$ with partial ordering $\sqsubseteq$. Pre-orders are assumed to be turned into partial orderings by considering the quotient by the preorder equivalence so $A^{\Sigma}_{\mathcal{I}}$ is a poset but may be not a complete lattice nor a cpo.

The meaning of the abstract properties is defined by an increasing *concretization* function $\gamma^{\Sigma}_{\mathcal{I}} \in A^{\Sigma}_{\mathcal{I}} \xrightarrow{\nearrow} \mathfrak{P}^{\Sigma}_{\mathcal{I}}$ [8]. In case of existence of a best abstraction, we use a *Galois connection* $\langle \mathfrak{P}^{\Sigma}_{\mathcal{I}}, \subseteq \rangle \underset{\alpha^{\Sigma}_{\mathcal{I}}}{\overset{\gamma^{\Sigma}_{\mathcal{I}}}{\leftrightarrows}} \langle A^{\Sigma}_{\mathcal{I}}, \sqsubseteq \rangle$ [7] such that, by definition, $\forall P \in \mathfrak{P}^{\Sigma}_{\mathcal{I}}, \overline{P} \in A^{\Sigma}_{\mathcal{I}}$: $\alpha^{\Sigma}_{\mathcal{I}}(P) \sqsubseteq \overline{P}$ if and only if $P \subseteq \gamma^{\Sigma}_{\mathcal{I}}(\overline{P})$.

The soundness of abstract domains $\langle A^{\Sigma}_{\mathcal{I}}, \sqsubseteq \rangle$, is defined as, for all $\overline{P}, \overline{Q} \in A^{\Sigma}_{\mathcal{I}}$,

$$\begin{array}{ll}
(\overline{P} \sqsubseteq \overline{Q}) \Rightarrow (\gamma^{\Sigma}_{\mathcal{I}}(\overline{P}) \subseteq \gamma^{\Sigma}_{\mathcal{I}}(\overline{Q})) & \text{implication} \\
\gamma^{\Sigma}_{\mathcal{I}}(\top) = \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}^{\Sigma}_{I} \right\} & \text{supremum} \\
\gamma^{\Sigma}_{\mathcal{I}}(\overline{P} \sqcup \overline{Q}) \supseteq (\gamma^{\Sigma}_{\mathcal{I}}(\overline{P}) \cup \gamma^{\Sigma}_{\mathcal{I}}(\overline{Q})) & \text{join} \\
\ldots &
\end{array}$$

The concrete least fixpoint semantics $C^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!]$ of a program P in Sect. **2.5** may have no correspondent in the abstract e.g. because the abstract domain $\langle A^{\Sigma}_{\mathcal{I}}, \sqsubseteq \rangle$ is not a cpo so

that the abstract transformer has no least fixpoint, even maybe no fixpoint. In that case, we can define the abstract semantics as the set of abstract inductive invariants of the program P.

$$\begin{array}{lcll} \overline{C}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] & \in & \wp(A^{\Sigma}_{\mathcal{I}}) & \text{abstract semantics} \\ \overline{F}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] & \in & A^{\Sigma}_{\mathcal{I}} \to A^{\Sigma}_{\mathcal{I}} & \text{abstract transformer} \\ \overline{C}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] & \triangleq & \left\{ \overline{P} \;\middle|\; \overline{F}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!](\overline{P}) \sqsubseteq \overline{P} \right\} & \text{postfixpoint semantics.} \end{array}$$

In practice, only one abstract postfixpoint needs to be computed (while the abstract semantics defines all possible ones). Such an abstract postfixpoint can be computed e.g. by elimination or iteratively from the infimum using widenings and narrowings [9].

In the concrete semantics the least fixpoint is, by Tarski's theorem, an equivalent representation of the set of concrete postfixpoints [28].

2.7 Soundness and completeness of abstract semantics

The abstract semantics $\overline{C}[\![\mathtt{P}]\!] \in A$ is *sound* with respect to a concrete semantics $C[\![\mathtt{P}]\!]$ of a program P for concretization γ whenever $\forall \overline{P} \in A : (\exists \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \overline{C} \sqsubseteq \overline{P}) \Rightarrow (C[\![\mathtt{P}]\!] \subseteq \gamma(\overline{P}))$. It is *complete* whenever $\forall \overline{P} \in A : (C[\![\mathtt{P}]\!] \subseteq \gamma(\overline{P})) \Rightarrow (\exists \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \overline{C} \sqsubseteq \overline{P})$. When the concrete semantics is defined in fixpoint form $C[\![\mathtt{P}]\!] \triangleq \mathbf{lfp}^{\subseteq} F[\![\mathtt{P}]\!]$ and the abstract semantics in postfixpoints, the soundness of the abstract semantics follows from the soundness conditions of the abstraction in Sect. **2.6** and the soundness of the abstract transformer [6, 7]

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

If the concrete semantics is also defined in postfixpoint form, then the soundness condition becomes

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

Moreover, the composition of sound abstractions is necessarily sound. To prove this let $\overline{C}[\![\mathtt{P}]\!]$ be a sound abstraction of $C[\![\mathtt{P}]\!]$ (on an abstract domain $\langle A_1, \sqsubseteq_1, \ldots \rangle$ with concretization γ_1), and $\overline{\overline{C}}[\mathtt{P}]$ be a sound abstraction of $\overline{C}[\![\mathtt{P}]\!]$ (on an abstract domain $\langle A_2, \sqsubseteq_2, \ldots \rangle$ with concretization γ_2), then $\forall \overline{\overline{P}} \in A_2$:

$$\begin{array}{ll} & \exists \overline{\overline{C}} \in \overline{\overline{C}}[\mathtt{P}] : \overline{\overline{C}} \sqsubseteq_2 \overline{\overline{P}} \\ \Rightarrow & \exists \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \overline{C} \sqsubseteq_1 \gamma_2(\overline{\overline{P}}) \qquad \text{⟅soundness of } \overline{\overline{C}}[\mathtt{P}]\text{⟆} \\ \Rightarrow & \exists C \in C[\![\mathtt{P}]\!] : C \subseteq \gamma_1(\gamma_2(\overline{\overline{P}})) \\ & \qquad \text{⟅soundness of } \overline{C}[\![\mathtt{P}]\!] \text{ choosing } \overline{P} = \gamma_2(\overline{\overline{P}}) \,.\text{⟆} \end{array}$$

The soundness of $\overline{F}[\![\mathtt{P}]\!]$ can usually be proved by induction on the syntactical structure of the program P using local soundness conditions.

$\gamma(\bar{\mathsf{f}}[\![\mathtt{x} := t]\!]\overline{P}) \supseteq \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := t]\!]\gamma(\overline{P})$ assignment postcondition
$\gamma(\bar{\mathsf{b}}[\![\mathtt{x} := t]\!]\overline{P}) \supseteq \mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := t]\!]\gamma(\overline{P})$ assignment precondition
$\gamma(\bar{\mathsf{p}}[\![\varphi]\!]\overline{P}) \supseteq \mathsf{p}_{\mathcal{I}}[\![\varphi]\!]\gamma(\overline{P})$ test.

2.8 Abstractions between multi-interpretations

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

If we can only compute properties on the standard interpretation $\mathfrak{I}$ 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 by the standard semantics. So, in general, we have two sets of interpretations, one is $\boldsymbol{\mathcal{I}}$, the context of interpretations for the program and the other one is $\boldsymbol{\mathcal{I}}^{\sharp}$, the set of interpretations used in the analysis. The relation between the two forms a Galois connection $\langle \wp^{\Sigma}_{\boldsymbol{\mathcal{I}}}, \subseteq \rangle \underset{\alpha^{\Sigma}_{\boldsymbol{\mathcal{I}} \to \boldsymbol{\mathcal{I}}^{\sharp}}}{\overset{\gamma^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp} \to \boldsymbol{\mathcal{I}}}}{\leftrightarrows}} \langle \wp^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp}}, \subseteq \rangle$ where

$$\begin{array}{lcl} \alpha^{\Sigma}_{\boldsymbol{\mathcal{I}} \to \boldsymbol{\mathcal{I}}^{\sharp}}(P) & \triangleq & P \cap \mathfrak{R}^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp}} \\ \gamma^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp} \to \boldsymbol{\mathcal{I}}}(Q) & \triangleq & \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma}_{\mathcal{I}} \;\middle|\; I \in \boldsymbol{\mathcal{I}}^{\sharp} \Rightarrow \langle I, \eta \rangle \in Q \right\} . \end{array}$$

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

$$\begin{array}{lll} & \alpha^{\Sigma}_{\boldsymbol{\mathcal{I}} \to \boldsymbol{\mathcal{I}}^{\sharp}}(P) \subseteq Q & \\ \Leftrightarrow & \forall \langle I, \eta \rangle \in P \cap \mathfrak{R}^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp}} : \langle I, \eta \rangle \in Q & \text{〈def. } \alpha^{\Sigma}_{\boldsymbol{\mathcal{I}} \to \boldsymbol{\mathcal{I}}^{\sharp}} \text{ and } \subseteq\text{〉} \\ \Leftrightarrow & \forall \langle I, \eta \rangle \in P : \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp}} \Rightarrow \langle I, \eta \rangle \in Q & \text{〈def. } \cap\text{〉} \\ \Leftrightarrow & \forall \langle I, \eta \rangle \in P : \langle I, \eta \rangle \in \boldsymbol{\mathcal{I}}^{\sharp} \Rightarrow \langle I, \eta \rangle \in Q & \\ & \quad \text{〈} \langle I, \eta \rangle \in P \in \wp^{\Sigma}_{\mathcal{I}} = \wp(\mathfrak{R}^{\Sigma}_{\mathcal{I}}) \text{ so } \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma}_{\mathcal{I}} \text{ proving } \eta \in \mathcal{R}^{\Sigma}_{I} \text{ hence, under these} & \\ & \quad \text{conditions, } I \in \boldsymbol{\mathcal{I}}^{\sharp} \text{ iff } \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp}} \text{〉} & \\ \Leftrightarrow & P \subseteq \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma}_{\mathcal{I}} \;\middle|\; I \in \boldsymbol{\mathcal{I}}^{\sharp} \Rightarrow \langle I, \eta \rangle \in Q \right\} & \text{〈def. } \subseteq\text{〉} \\ \Leftrightarrow & P \subseteq \gamma^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp} \to \boldsymbol{\mathcal{I}}}(Q) & \text{〈def. } \gamma^{\Sigma}_{\boldsymbol{\mathcal{I}}^{\sharp} \to \boldsymbol{\mathcal{I}}} \text{.〉} \quad \square \end{array}$$

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

2.9 Soundness of the multi-interpretation abstract transformers

For all $P^\sharp \in \mathfrak{P}^\Sigma_{\mathcal{I}^\sharp}$, we have soundness

$$
\begin{aligned}
& \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!] \circ \gamma^\Sigma_{\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^\Sigma_{\mathcal{I}^\sharp \to \mathcal{I}}(P^\sharp) \right\} \\
& \qquad\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|\; \langle I,\, \eta \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta \rangle \in P^\sharp \right) \right\} \\
& \qquad\qquad \wr \text{def. } \gamma^\Sigma_{\mathcal{I}^\sharp \to \mathcal{I}} \int \\
\subseteq \; & \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \;\middle|\; I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta \rangle \in P^\sharp \right\} \\
& \qquad\qquad \wr \text{since } \langle I,\, \eta \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \text{ implies } \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \int \\
= \; & \left\{ \langle I,\, \eta' \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \;\middle|\; I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta' \rangle \in \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \;\middle|\; \langle I,\, \eta \rangle \in P^\sharp \right\} \right\} \\
& \qquad\qquad \wr \text{letting } \eta' = \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta \in \mathfrak{R}^\Sigma_{\mathcal{I}}] \int \\
= \; & \left\{ \langle I,\, \eta' \rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \;\middle|\; I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta' \rangle \in \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp) \right\} \qquad \wr \text{def. } \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!] \int \\
\subseteq \; & \gamma^\Sigma_{\mathcal{I}^\sharp \to \mathcal{I}} \circ \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp) \qquad \wr \text{def. } \gamma^\Sigma_{\mathcal{I}^\sharp \to \mathcal{I}} \int
\end{aligned}
$$

and similarly for the other transformers.

2.10 Uniform abstraction of interpretations

We may want to 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 possible interpretations of the program. That abstraction simply forgets the interpretations and just keeps the union of all possible behaviors. The abstraction is described by the Galois connection $\langle \mathfrak{P}^\Sigma_{\mathcal{I}},\, \subseteq \rangle \underset{\overline{\alpha}^\Sigma_{\mathcal{I}}}{\overset{\overline{\gamma}^\Sigma_{\mathcal{I}}}{\leftrightarrows}} \langle \bigcup_{I \in \mathcal{I}} \mathcal{R}^\Sigma_I,\, \subseteq \rangle$ where

$$\overline{\alpha}^\Sigma_{\mathcal{I}}(P) \triangleq \left\{ \eta \;\middle|\; \exists I : \langle I,\, \eta \rangle \in P \right\} \text{ and } \overline{\gamma}^\Sigma_{\mathcal{I}}(E) \triangleq \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in E \right\}.$$

Example 2. That is what the Astrée analyzer [1, 10] does when taking into account all possible rounding error modes for floating points computations. □

2.11 Theories

The set $\vec{\mathtt{x}}_\Psi$ of *free variables* of a formula $\Psi \in \mathbb{F}(\Sigma)$ is defined inductively as the set of variables in the formula which are not in the scope of an existential quantifier. A *sentence* of $\mathbb{F}(\Sigma)$ is a formula with no free variable, $\mathbb{S}(\Sigma) \triangleq \left\{ \Psi \in \mathbb{F}(\Sigma) \;\middle|\; \vec{\mathtt{x}}_\Psi = \emptyset \right\}$. A *theory* $\mathcal{T} \in \wp(\mathbb{S}(\Sigma))$ is a set of sentences [4] (called the *theorems* of the theory). The set of predicate and function symbols that appear in at least one sentence of a theory $\mathcal{T}$ should be contained in the *signature* $\mathbb{S}(\mathcal{T}) \subseteq \Sigma$ of theory $\mathcal{T}$. The *language* $\mathbb{F}(\mathcal{T})$ of a theory $\mathcal{T}$ 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.

2.12 Models

An interpretation $I \in \mathfrak{I}(\Sigma)$ is said to be a *model* of $\Psi \in \mathbb{F}(\Sigma)$ when $\exists \eta : I \models_\eta \Psi$ (i.e. I makes Ψ true). An interpretation is a *model* of a theory $\mathcal{T}$ if and only if 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

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

since if Ψ is a sentence and if there is an I and an 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 or enumerable) set of axioms which generate the set of theorems of the theory by implication. A theory is said to be *deductive* if and only if it is closed by deduction, that is all the theorems that are true on all models of the theory are in the theory.

2.13 Abstraction by a theory

Another possibility 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 interpretation, environment. In some cases it can be difficult to represent exactly an infinite set of interpretations, and we can use theories (preferably deductive theories 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

$$\gamma_{\mathfrak{M}}(\mathcal{T}) \triangleq \left\{ \langle I, \eta \rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge \eta \in \mathcal{R}_I^\Sigma \right\} .$$

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

Example 3. If $\mathfrak{I}$ interprets programs over the natural numbers, then by Gödel 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 $\boldsymbol{\mathcal{I}}$ of interpretations there is a best abstraction $\alpha^{\mathbb{S}(\mathcal{T})}_{\boldsymbol{\mathcal{I}} \to \gamma_{\mathfrak{M}}(\mathcal{T})}(P)$ of interpreted properties in $P \in \mathfrak{P}^{\mathbb{S}(\mathcal{T})}_{\boldsymbol{\mathcal{I}}}$ by abstract properties in $\mathfrak{P}^{\mathbb{S}(\mathcal{T})}_{\gamma_{\mathfrak{M}}(\mathcal{T})}$.

2.14 Logical abstract domains

A *logical abstract domain* is an abstract domain $\langle A^\Sigma_\mathcal{T}, \sqsubseteq, \top, \sqcup, \ldots, \bar{\mathtt{f}}, \overline{\mathtt{p}}, \ldots\rangle$ such that $\mathcal{T} \in \wp(\mathbb{S}(\Sigma))$ and $A^\Sigma_\mathcal{I} \subseteq \mathbb{F}(\mathcal{T})$ with $\sqsubseteq \triangleq \Rightarrow$, $\top \triangleq \mathtt{tt}$, $\sqcup \triangleq \vee$, etc, and the concretization is $\gamma^\Sigma_\mathcal{T}(\Psi) \triangleq \left\{ \langle I, \eta\rangle \in \gamma_\mathfrak{M}(\mathcal{T}) \;\middle|\; I \models_\eta \Psi \right\}$.

Remark that there might be no finite formula in the language $\mathbb{F}(\mathcal{T})$ of the theory $\mathcal{T}$ to encode a best abstraction. In absence of a best abstraction there is no Galois connection, in which case soundness can always be formalized by a concretization function only as in Sect. **2.6**. Moreover, in presence of infinite ascending chains of finite first-order formulæ (e.g. $(\mathtt{x} = 0) \Rightarrow (\mathtt{x} = 0 \vee \mathtt{x} = 1) \Rightarrow \ldots \Rightarrow \bigvee_{i=1}^n \mathtt{x} = i \Rightarrow \ldots$) and descending chains of finite formulæ (e.g. $(\mathtt{x} \neq -1) \Leftarrow (\mathtt{x} \neq -1 \wedge \mathtt{x} \neq -2) \Leftarrow \ldots \Leftarrow \bigwedge_{i=1}^n \mathtt{x} \neq -i \Leftarrow \ldots$) with no finite first-order formula to express their limits, the fixpoint may not exist. Hence the fixpoint semantics in the style of Sect. **2.5** is not well-defined in the abstract. However, following Sect. **2.6**, we can define the abstract semantics as the set of abstract inductive invariants

$$
\begin{array}{lcll}
\overline{C}^{\mathbb{S}(\mathcal{T})}_\mathcal{T}[\![\mathtt{P}]\!] & \in & \wp(\mathbb{F}(\mathcal{T})) & \text{abstract semantics} \\
\overline{F}^{\mathbb{S}(\mathcal{T})}_\mathcal{T}[\![\mathtt{P}]\!] & \in & \mathbb{F}(\mathcal{T}) \stackrel{\cdot}{\to} \mathbb{F}(\mathcal{T}) & \text{abstract transformer} \\
\overline{C}^{\mathbb{S}(\mathcal{T})}_\mathcal{T}[\![\mathtt{P}]\!] & \triangleq & \left\{ \overline{P} \;\middle|\; \overline{F}^{\mathbb{S}(\mathcal{T})}_\mathcal{T}[\![\mathtt{P}]\!](\overline{P}) \subseteq \overline{P} \right\} & \text{postfixpoint semantics.}
\end{array}
$$

3 Observational semantics

Besides values of program variables, it may be interesting to allow the concrete semantics to observe values of auxiliary variables (e.g. as in Owicki and Gries proof method for parallel processes [24] which does not use the program counters and so requires using auxiliary variables) or values of functions over program variables (such as wait conditions in monitors [20]). Whereas such cases can be described in the general setting above (e.g. by inclusion of the auxiliary variables as program variables in Owicki and Gries proof method or by reevaluating wait conditions if a signal on the sensitivity lists changes), it is more convenient to explicitly define the observables of the program semantics.

3.1 Observable properties of multi-interpreted programs

We name observables by identifiers (which, in particular, can be variable identifiers). Observables are functions from values of program variables to values.

$$
\begin{array}{rcll}
\Sigma & = & \langle \mathbb{s}, \mathbb{x}, \mathbb{f}, \mathbb{p}, \# \rangle & \text{signature} \\
\mathtt{x} & \in & \mathbb{x}_\mathtt{P} & \text{program variables } (\mathbb{x}_\mathtt{P} \subseteq \mathbb{x}) \\
\Sigma_\mathtt{P} & = & \langle \mathbb{s}, \mathbb{x}_\mathtt{P}, \mathbb{f}, \mathbb{p}, \# \rangle \mathrel{\dot\subseteq} \Sigma & \text{program signature} \\
x & \in & \mathbb{x}_\mathcal{O} & \text{observable identifiers } (\mathbb{x}_\mathcal{O} \subseteq \mathbb{x}) \\
\Sigma_\mathcal{O} & = & \langle \mathbb{s}, \mathbb{x}_\mathcal{O}, \mathbb{f}, \mathbb{p}, \# \rangle \mathrel{\dot\subseteq} \Sigma & \text{observable signature} \\
v & \in & I_\mathcal{V} & \text{values (for interpretation } I \in \mathfrak{I}(\Sigma)) \\
\eta & \in & \mathcal{R}^{\Sigma_\mathtt{P}}_I \triangleq \mathbb{x}_\mathtt{P} \to I_\mathcal{V} & \text{program variable environments}
\end{array}
$$

$$
\begin{array}{rcll}
\mathcal{I} & \in & \wp(\mathfrak{I}(\Sigma)) & \text{multiple interpretations} \\
\mathfrak{R}_{\mathcal{I}}^{\Sigma_P} & \triangleq & \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I^{\Sigma_P} \right\} & \text{multi-interpreted progr. envir.} \\
\mathfrak{P}_{\mathcal{I}}^{\Sigma_P} & \triangleq & \wp(\mathfrak{R}_{\mathcal{I}}^{\Sigma_P}) & \text{multi-interpreted program properties} \\
\zeta & \in & \mathcal{R}_I^{\Sigma_O} \triangleq \mathbb{X}_O \to I_{\mathcal{V}} & \text{program observable environments} \\
\mathfrak{R}_{\mathcal{I}}^{\Sigma_O} & \triangleq & \left\{ \langle I,\ \zeta\rangle \;\middle|\; I \in \mathcal{I} \wedge \zeta \in \mathcal{R}_I^{\Sigma_O} \right\} & \text{multi-interpreted environments} \\
\mathfrak{P}_{\mathcal{I}}^{\Sigma_O} & \triangleq & \wp(\mathfrak{R}_{\mathcal{I}}^{\Sigma_O}) & \text{multi-interpreted observable properties} \\
\omega_I & \in & \mathcal{O}_I^{\Sigma_P} \triangleq \mathcal{R}_I^{\Sigma_P} \to I_{\mathcal{V}} & \text{observables (for } I \in \mathcal{I}) \\
\Omega_I & \in & \mathbb{X}_O \to \mathcal{O}_I^{\Sigma_P} & \text{observable naming.}
\end{array}
$$

Whereas a concrete or abstract *program semantics* is relative to $\mathfrak{P}_{\mathcal{I}}^{\Sigma_P}$ or $A_{\mathcal{I}}^{\Sigma_P}$, the *observational semantics* is relative to $\mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ or $A_{\mathcal{I}}^{\Sigma_O}$ and both can be specified in fixpoint or in postfixpoint form.

Example 4 (Variable multiple observations). We may want to observe values of variables at different time instants, as abstracted to program points, which leads to SSA [12] renaming variables such that a property of static single assignment holds. □

Example 5 (Term observation). We may want to observe values of a term t of the program for a particular interpretation I stored in an auxiliary variable $x \in \mathbb{X}_O \setminus \mathbb{X}_P$ so that $\Omega_I(x) \triangleq [\![t]\!]_I$. For example, quaternions are analyzed in [1] by observing the value of their norm $t = \sqrt{\mathtt{a}^2 + \mathtt{b}^2 + \mathtt{c}^2 + \mathtt{d}^2}$ (where $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d} \in \mathbb{X}_P \cup \mathbb{X}_O$ are variables) for several rounding semantics of floats. □

Example 6 (Combining symbolic and numerical analyzes). One can observe the length of a list, the height of a stack, etc. and reuse classical numerical abstractions on that observation [13]. □

Example 7 (Memory model). In the memory model of [21], a 32 bits unsigned/positive integer variable $\mathtt{x}$ can be encoded by its constituent bytes $\langle x_3,\ x_2,\ x_1,\ x_0\rangle$ so that, for little endianness, $\eta(\mathtt{x}) = \Omega_I(x_3)\eta \times 2^{32} + \Omega_I(x_2)\eta \times 2^{16} + \Omega_I(x_1)\eta \times 2^8 + \Omega_I(x_0)\eta$. □

Given a program property $P \in \mathfrak{P}_{\mathcal{I}}^{\Sigma_P}$, the corresponding observable property is

$$\alpha_{\mathcal{I}}^{\Omega}(P) \triangleq \left\{ \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_O} \;\middle|\; \langle I,\ \eta\rangle \in P \right\}.$$

$\zeta(x)$ is therefore the value $\Omega_I(x)\eta$ of the observable named x when the values of program variables are given by η. Inversely, given an observable property $Q \in \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$, the corresponding program property is

$$\gamma_{\mathcal{I}}^{\Omega}(Q) \triangleq \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_P} \;\middle|\; \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in Q \right\}.$$

Example 8 (Variable observation). If we limit our observations to values of program variables then $\mathbb{X}_O = \mathbb{X}_P$ and $\forall \mathtt{x} \in \mathbb{X}_O : \forall \eta \in \mathcal{R}_I^{\Sigma_P} : \Omega_I(\mathtt{x})\eta \triangleq \eta(\mathtt{x})$ so that $\lambda\mathtt{x}.\Omega_I(\mathtt{x})\eta = \eta$ pointwise hence $\lambda\, \mathtt{x} \bullet \Omega_I(\mathtt{x})$, $\alpha_{\mathcal{I}}^{\Omega}$ and $\gamma_{\mathcal{I}}^{\Omega}$ are the identity. If we do not want to analyze the values of all program variables then $\mathbb{X}_O \subsetneq \mathbb{X}_P$. □

We have the Galois connection

Theorem 9. $\langle \mathfrak{P}_I^{\Sigma_P},\ \subseteq\rangle \underset{\alpha_I^\Omega}{\overset{\gamma_I^\Omega}{\leftrightarrows}} \langle \mathfrak{P}_I^{\Sigma_O},\ \subseteq\rangle$.

Proof.

$$
\begin{array}{ll}
& \alpha_I^\Omega(P) \subseteq Q \\
\Leftrightarrow & \left\{ \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \mathfrak{R}_I^{\Sigma_O} \;\middle|\; \langle I,\ \eta\rangle \in P \right\} \subseteq Q \qquad \wr \text{def. } \alpha_I^\Omega \int \\
\Leftrightarrow & \forall \langle I,\ \eta\rangle \in P : \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in Q \qquad \wr \text{def. } \subseteq \text{ and } Q \in \mathfrak{P}_I^{\Sigma_O} = \wp(\mathfrak{R}_I^{\Sigma_O}) \int \\
\Leftrightarrow & P \subseteq \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \;\middle|\; \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in Q \right\} \qquad \wr \text{def. } \subseteq \text{ and } P \subseteq \mathfrak{R}_I^{\Sigma_P} \int \\
& P \subseteq \gamma_I^\Omega(Q) \qquad \wr \text{def. } \gamma_I^\Omega \,. \int \quad \square
\end{array}
$$

3.2 Soundness of the abstraction of observable properties

The *observational abstraction* will be of observable properties in $\mathfrak{P}_I^{\Sigma_O}$ so with concretization $\gamma_I^{\Sigma_O} \in A_I^{\Sigma_O} \to \mathfrak{P}_I^{\Sigma_O}$ where $A_I^{\Sigma_O}$ is the abstract domain. The classical direct abstraction of program properties in $\mathfrak{P}_I^{\Sigma_P}$ will be the particular case where $\mathbb{x}_O = \mathbb{x}_P$ and $\lambda\, \mathtt{x} \bullet \Omega_I(\mathtt{x})$ is the identity. The program properties corresponding to observable Ω_I are given by

$$
\begin{array}{l}
\boldsymbol{\gamma}_I^{\Omega,P} \in A_I^{\Sigma_O} \mapsto \mathfrak{P}_I^{\Sigma_P} \\
\boldsymbol{\gamma}_I^{\Omega,P} \triangleq \boldsymbol{\gamma}_I^{\Omega} \circ \gamma_I^{\Sigma_O} = \lambda\, \overline{P} \bullet \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \;\middle|\; \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\overline{P}) \right\}.
\end{array}
$$

Under the observational semantics, soundness conditions remain unchanged, but they must be proved with respect to $\boldsymbol{\gamma}_I^{\Omega,P}$, not $\gamma_I^{\Sigma_O}$. So the soundness conditions on transformers become slightly different. For example the soundness condition on the abstract postcondition $\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]$ becomes:

$$
\begin{array}{ll}
& \boldsymbol{\gamma}_I^{\Omega,P}(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}) \quad \supseteq \quad \mathtt{f}_I[\![\mathtt{x} := e]\!](\boldsymbol{\gamma}_I^{\Omega,P}(\overline{P})) \\
\Leftrightarrow & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \;\middle|\; \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}) \right\} \quad \supseteq \\
& \quad \left\{ \langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \;\middle|\; \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \wedge \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\overline{P}) \right\} \\
& \qquad\qquad \wr \text{def. } \boldsymbol{\gamma}_I^{\Omega,P}, \mathtt{f}_I[\![\mathtt{x} := e]\!], \text{ and } \boldsymbol{\gamma}_I^{\Omega,P} \int \\
\Leftrightarrow & \forall \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} : \forall \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\overline{P}) : \\
& \quad \langle I,\ \lambda\, x \bullet \Omega_I(x)(\eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta])\rangle \in \gamma_I^{\Sigma_O}\left(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}\right) \qquad \wr \text{def. } \subseteq \int \\
\Leftrightarrow & \gamma_I^{\Sigma_O}\left(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}\right) \supseteq \left\{ \langle I,\ \lambda\, x \bullet \Omega_I(x)(\eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta])\rangle \;\middle|\; \langle I,\ \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \wedge \langle I,\ \lambda\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\overline{P}) \right\} \\
& \qquad\qquad \wr \text{def. } \subseteq \,. \int
\end{array}
$$

3.3 Observational extension

It can sometimes be useful to extend an abstract property $\overline{P}$ for observables Ω with a new observable ω named x. We will write $\mathsf{extend}_{(x,\omega)}\left(\overline{P}\right)$ for the extension of $\overline{P}$ with observable ω for observable identifier x.

Example 10. Let $A_{\mathbb{x}_O}$ be the abstract domain mapping observable identifiers $\mathtt{x} \in \mathbb{x}_O$ to an interval of values [6]. Assume that intervals of program variables are observable, that is $\mathbb{x}_P \subseteq \mathbb{x}_O$ and let $\mathtt{x} \in \mathbb{x}_P$ be a program variables for which we want to observe the square $\mathtt{x}^2$ so $\omega_I \triangleq [\![\mathtt{x}^2]\!]_I$. Let $\mathtt{x2} \notin \mathbb{x}_O$ be a fresh name for this observable. This extension of observable properties with a new observable can be defined as

$$\begin{array}{rcl} \mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])} & \in & A_{\mathbb{x}_O} \to A_{\mathbb{x}_O \cup \{\mathtt{x2}\}} \\ \mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])}\left(\overline{P}\right) & \triangleq & \lambda\, x \in \mathbb{x}_O \cup \{\mathtt{x2}\} \bullet (x \neq \mathtt{x2}\ ?\ \overline{P}(x)\ : \\ & & \qquad \overline{P}(\mathtt{x2}) \otimes \overline{P}(\mathtt{x2})\ ) \end{array}$$

(where $\otimes$ is the product of intervals) is sound. □

The semantics of the extension operation is defined by its soundness condition

$$\boldsymbol{\gamma}_{\mathcal{I}}^{\lambda I \bullet\ \lambda y \bullet y = x\ ?\ \omega_I : \Omega_I(y), \mathrm{P}}\left(\mathsf{extend}_{(x,\omega)}\left(\overline{P}\right)\right) \quad \supseteq \quad \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}\left(\overline{P}\right)\ .$$

Example 11. For the logical abstract domain $A \triangleq \mathbb{F}(\Sigma)$ with $\gamma_{\mathcal{I}}^{\Sigma_O}(\Psi) \triangleq \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge I \models_\eta \Psi \right\}$,

$$\mathsf{extend}_{(x,[\![e]\!])}(\Psi[x \leftarrow e]) \quad \triangleq \quad \exists\, x : (x = e \wedge \Psi) \quad \text{is sound.}$$

Proof.

$$\begin{array}{ll} & \boldsymbol{\gamma}_{\mathcal{I}}^{\lambda I \bullet\ \lambda y \bullet y = x\ ?\ [\![e]\!]_I : \Omega_I(y), \mathrm{P}}\left(\mathsf{extend}_{(x,[\![e]\!])}(\Psi[x \leftarrow e])\right) \\ = & \boldsymbol{\gamma}_{\mathcal{I}}^{\lambda I \bullet\ \lambda y \bullet y = x\ ?\ [\![e]\!]_I : \Omega_I(y), \mathrm{P}}(\exists\, x : (x = e \wedge \Psi)) \\ & \qquad\qquad \wr \text{def. } \mathsf{extend}_{(x,[\![e]\!])}(\Psi[x \leftarrow e]) \triangleq \exists\, x : (x = e \wedge \Psi) \wr \\ = & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I,\ \lambda\, x' \bullet (\lambda\, y \bullet y = x\ ?\ [\![e]\!]_I\ :\ \Omega_I(y))\,(x')\eta\rangle \in \gamma_{\mathcal{I}}^{\Sigma_O}(\exists\, x : (x = e \wedge \Psi)) \right\} \quad \wr \text{def. } \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}} \wr \\ = & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I,\ \lambda\, x' \bullet (x' = x\ ?\ [\![e]\!]_I(\eta)\ :\ \Omega_I(x')\eta)\rangle \in \gamma_{\mathcal{I}}^{\Sigma_O}(\exists\, x : (x = e \wedge \Psi)) \right\} \quad \wr \text{def. application} \wr \\ = & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_\mathrm{P}} \;\middle|\; I \in \mathcal{I} \wedge I \models_{\lambda x' \bullet (x' = x\ ?\ [\![e]\!]_I(\eta) : \Omega_I(x')\eta)} \exists\, x : (x = e \wedge \Psi) \right\} \\ & \qquad\qquad \wr \text{def. } \gamma_{\mathcal{I}}^{\Sigma_O}(\Psi) \triangleq \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge I \models_\eta \Psi \right\} \wr \\ = & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_\mathrm{P}} \;\middle|\; I \in \mathcal{I} \wedge I \models_{\lambda x' \bullet \Omega_I(x')\eta} \Psi[x \leftarrow e] \right\} \quad \wr \text{def. substitution} \wr \\ = & \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I,\ \lambda\, x' \bullet \Omega_I(x')\eta\rangle \in \gamma_{\mathcal{I}}^{\Sigma_O}(\Psi[x \leftarrow e]) \right\} \quad \wr \text{def. } \gamma_{\mathcal{I}}^{\Sigma_O} \wr \\ = & \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}(\Psi[x \leftarrow e]) \quad \wr \text{def. } \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}} \wr \quad \square\ \square \end{array}$$

This extension operation can also be used for vectors of fresh variables and vectors of observables in the natural way.

Definition 12 (Extension of observable properties with new observables). *Let $A_{\mathcal{I}}^{\Sigma_O}$ be an abstract domain with partial ordering $\sqsubseteq$ abstracting multi-interpreted properties in $\mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ for signature Σ_O with observable identifiers $\mathbb{x}_O \subseteq \mathbb{x}$, multi-interpretations $\mathcal{I}$, and observables named by Ω such that $\forall I \in \mathcal{I} : \Omega_I \in \mathbb{x}_O \to O_I^{\Sigma_\mathrm{P}}$.*

Consider the new observables Ω' such that $\forall I \in \mathcal{I} : \Omega'_I \in \mathbb{x}_{O'} \setminus \mathbb{x}_O \to O_I^{\Sigma_\mathrm{P}}$ where $\mathbb{x}_{O'}$ are the new observable names such that $\mathbb{x}_O \subseteq \mathbb{x}_{O'}$. The abstraction now uses the abstract domain $A_{\mathcal{I}}^{\Sigma_{O'}}$ with partial ordering $\sqsubseteq'$ abstracting multi-interpreted properties in $\mathfrak{P}_{\mathcal{I}}^{\Sigma_{O'}}$ for signature $\Sigma_{O'}$ with observable identifiers $\mathbb{x}_{O'} \subseteq \mathbb{x}$. A sound extension $\mathsf{extend}_{\Omega'} \in A_{\mathcal{I}}^{\Sigma_O} \to A_{\mathcal{I}}^{\Sigma_{O'}}$ *satisfies the soundness condition*

$$\boldsymbol{\gamma}_{\mathcal{I}}^{\lambda I \bullet \lambda y \bullet y \in \mathbb{X}_{\mathcal{O}'} \setminus \mathbb{X}_{\mathcal{O}} ? \Omega'_I(y) : \Omega_I(y), \mathrm{P}}\left(\mathsf{extend}_{\Omega'}\left(\overline{P}\right)\right) \quad \supseteq \quad \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega, \mathrm{P}}\left(\overline{P}\right) \ .$$

Given $A \subseteq A_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}$, we write $\mathsf{extend}_{\Omega'}\,[A] \triangleq \left\{ \mathsf{extend}_{\Omega'}\left(\overline{P}\right) \;\middle|\; \overline{P} \in A \right\}$.

4 Iterated reduction

A reduction makes a property more precise in the abstract without changing its concrete meaning. By iterating this reduction, one can improve the precision of a static analysis without altering its soundness.

Definition 13 (Reduction). *Let $\langle A, \sqsubseteq \rangle$ be a poset which is an abstract domain with concretization $\gamma \in A \xrightarrow{\nearrow} C$ where $\langle C, \subseteq \rangle$ is the concrete domain. A* reduction *is $\rho \in A \to A$ which is reductive that is $\forall \overline{P} \in A : \rho(\overline{P}) \sqsubseteq \overline{P}$ and sound in that $\forall \overline{P} \in A : \gamma(\rho(\overline{P})) = \gamma(\overline{P})$. The* iterates *of the reduction are $\rho^0 \triangleq \boldsymbol{\lambda}\, \overline{P} \bullet \overline{P}$, $\rho^{\lambda+1} = \rho(\rho^\lambda)$ for successor ordinals and $\rho^\lambda = \bigsqcap_{\beta<\lambda} \rho^\beta$ for limit ordinals. The iterates are* well-defined *when the greatest lower bounds $\bigsqcap$ (glb) do exist in the poset $\langle A, \sqsubseteq \rangle$.*

Lemma 14 (Iterated reduction). *Given a sound reduction ρ, for all ordinals λ, ρ^λ is a sound reduction. If the iterates of ρ from $\overline{P}$ are well-defined then their limit $\rho^*(\overline{P})$ exists. We have $\forall \beta < \lambda : \rho^*(\overline{P}) \sqsubseteq \rho^\lambda(\overline{P}) \sqsubseteq \rho^\beta(\overline{P}) \sqsubseteq \overline{P}$. If γ is the upper adjoint of a Galois connection then ρ^* is a sound reduction. It ρ is increasing then $\rho^* = \boldsymbol{\lambda}\, \overline{P} \bullet \boldsymbol{gfp}^{\sqsubseteq}_{\overline{P}} \rho$ is the greatest fixpoint (gfp) of ρ less than or equal to $\overline{P}$.*

Proof. Assuming the iterates of ρ from $\overline{P} \in A$ to be well-defined (otherwise the result holds only for $\lambda < \omega$, where ω is the first infinite ordinal) we observe, by transfinite induction, that the iterates form a descending chain since ρ is reductive and by definition of the glb $\bigsqcap$. By antisymmetry of $\sqsubseteq$ in the poset $\langle A, \sqsubseteq \rangle$, a fixpoint is reached when the ordinal ϵ has a cardinality greater than that of A since otherwise the iterates all contained in A would have a cardinality strictly greater than that of A. The iterates must be stationary beyond λ so that the limit $\rho^*(\overline{P}) \triangleq \rho^\epsilon(\overline{P})$ is well-defined. It follows that $\forall \beta < \lambda : \rho^*(\overline{P}) \sqsubseteq \rho^\lambda(\overline{P}) \sqsubseteq \rho^\beta(\overline{P}) \sqsubseteq \overline{P}$ since the iterates are $\sqsubseteq$-decreasing. Soundness follows by transfinite induction. For the basis $\gamma(\rho^0(\overline{P})) = \gamma(\overline{P})$ by def. of ρ^0. For successor ordinals, $\gamma(\rho^{\lambda+1}(\overline{P})) \triangleq \gamma(\rho(\rho^\lambda(\overline{P}))) = \gamma(\rho^\lambda(\overline{P})) = \gamma(\overline{P})$ since ρ is sound and by induction hypothesis. For limit ordinals λ, when γ is the upper adjoint of a Galois connection, hence preserves existing greatest lower bounds, we have $\rho^\lambda(\overline{P}) \triangleq \gamma(\bigsqcap_{\beta<\lambda} \rho^\beta(\overline{P})) = \bigsqcap_{\beta<\lambda} \gamma(\rho^\beta(\overline{P})) = \bigsqcap_{\beta<\lambda} \gamma(\overline{P}) = \gamma(\overline{P})$ by induction hypothesis and def. of glb. Let $\overline{Q}$ be another fixpoint of ρ assumed to be increasing such that $\overline{Q} \sqsubseteq \overline{P}$ so that $\forall \lambda : \overline{Q} = \rho^\lambda(\overline{Q}) \sqsubseteq \rho^\lambda(\overline{P})$ by transfinite induction proving that the limit $\overline{Q} \sqsubseteq \rho^\epsilon(\overline{P}) = \rho^*(\overline{P})$ is the gfp. □

A particular case of iterated reduction was proposed by [15] following [7].

5 Reduced product

5.1 Definition

Let $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$, Δ finite, be abstract domains with increasing concretization $\gamma_i \in A_i \to \mathfrak{P}^{\Sigma_O}_{\mathcal{I}}$. Their *Cartesian product* is $\langle \vec{A}, \vec{\sqsubseteq}\rangle$ where $\vec{A} \triangleq \bigtimes_{i\in\Delta} A_i$, $(\vec{P} \vec{\sqsubseteq} \vec{Q}) \triangleq \bigwedge_{i\in\Delta}(\vec{P}_i \sqsubseteq_i \vec{Q}_i)$ and $\vec{\gamma} \in \bigtimes_{i\in\Delta} A_i \to \mathfrak{P}^{\Sigma_O}_{\mathcal{I}}$ is $\vec{\gamma}(\vec{P}) \triangleq \bigcap_{i\in\Delta} \gamma_i(\vec{P}_i)$. In particular the *product* $\langle A_i \times A_j, \sqsubseteq_{ij}\rangle$ is such that $\langle x, y\rangle \sqsubseteq_{ij} \langle x', y'\rangle \triangleq (x \sqsubseteq_i x') \wedge (y \sqsubseteq_j y')$ and $\gamma_{ij}(\langle x, y\rangle) \triangleq \gamma_i(x) \cap \gamma_j(y)$.

Their *reduced product* is $\langle (\bigtimes_{i\in\Delta} A_i)/_{\vec{\equiv}}, \vec{\sqsubseteq}\rangle$ where $(\vec{P} \vec{\equiv} \vec{Q}) \triangleq (\vec{\gamma}(\vec{P}) = \vec{\gamma}(\vec{Q}))$ and $\vec{\gamma}$ as well as $\vec{\sqsubseteq}$ are naturally extended to the equivalence classes $[\vec{P}]/_{\vec{\equiv}}$, $\vec{P} \in \vec{A}$, of $\vec{\equiv}$ [7].

The simple cartesian product can be a representation for the reduced product, but if we just abstract transformers componentwise, then we obtain the same result as running analyses with each abstract domain independently. We can obtain much more precise results if we try to compute precise abstract values for each abstract domain, while staying in the same class of the reduced product. Computing such values is naturally a reduction.

5.2 Iterated product reduction

Implementations of the most precise reduction (if it exists) can hardly be modular since in general adding a new abstract domain to increase precision implies that the reduced product must be completely redesigned. On the contrary, the pairwise iterated product reduction below, is more modular, in that the introduction of a new abstract domain only requires defining the reduction with the other existing abstract domains.

For $i, j \in \Delta$, $i \neq j$, let $\rho_{ij} \in \langle A_i \times A_j, \sqsubseteq_{ij}\rangle \mapsto \langle A_i \times A_j, \sqsubseteq_{ij}\rangle$ be pairwise reductions (so that $\forall \langle x, y\rangle \in A_i \times A_j : \rho_{ij}(\langle x, y\rangle) \sqsubseteq_{ij} \langle x, y\rangle$, preferably lower closure operators i.e. reductive, increasing and idempotent). Define the pairwise reductions $\vec{\rho}_{ij} \in \langle \vec{A}, \vec{\sqsubseteq}\rangle \mapsto \langle \vec{A}, \vec{\sqsubseteq}\rangle$ of the Cartesian product as

$$\vec{\rho}_{ij}(\vec{P}) \quad \triangleq \quad \text{let } \langle \vec{P}'_i, \vec{P}'_j\rangle \triangleq \rho_{ij}(\langle \vec{P}_i, \vec{P}_j\rangle) \text{ in } \vec{P}[i \leftarrow \vec{P}'_i][j \leftarrow \vec{P}'_j]$$

where $\vec{P}[i \leftarrow x]_i = x$ and $\vec{P}[i \leftarrow x]_j = \vec{P}_j$ when $i \neq j$. Define the iterated pairwise reductions $\vec{\rho}^n$, $\vec{\rho}^* \in \langle \vec{A}, \vec{\sqsubseteq}\rangle \mapsto \langle \vec{A}, \vec{\sqsubseteq}\rangle$, $n \geqslant 0$ of the Cartesian product as in Def. **13** for

$$\vec{\rho} \quad \triangleq \quad \bigcirc_{\substack{i,j\in\Delta, \\ i\neq j}} \vec{\rho}_{ij} \tag{1}$$

where $\bigcirc_{i=1}^{n} f_i \triangleq f_{\pi_1} \circ \ldots \circ f_{\pi_n}$ is the function composition for some arbitrary permutation π of $[1, n]$.

The pairwise reductions $\vec{\rho}_{ij}$ and the iterated ones $\vec{\rho}^n$, $n \geqslant 0$ as well as their closure $\vec{\rho}^\star$, if any, are sound over-approximations of the reduced product in that

Theorem 15. *Under the hypotheses of Def. **13** and assuming the limit of the iterated reductions is well defined, the reductions are such that* $\forall \vec{P} \in \vec{A} : \forall \lambda : \vec{\rho}^\star(\vec{P}) \vec{\sqsubseteq} \vec{\rho}^\lambda(\vec{P}) \vec{\sqsubseteq} \vec{\rho}_{ij}(\vec{P}) \vec{\sqsubseteq} \vec{P}$*, $i, j \in \Delta$, $i \neq j$ and sound since* $\vec{\rho}^\lambda(\vec{P}), \vec{\rho}_{ij}(\vec{P}), \vec{P} \in [\vec{P}]/_{\vec{\equiv}}$ *and if γ preserves lower bounds,* $\vec{\rho}^\star(\vec{P}) \in [\vec{P}]/_{\vec{\equiv}}$ *also.*

Proof. The inequalities $\vec{\rho}^{\star}(\vec{P}) \vec{\sqsubseteq} \vec{\rho}^{\lambda}(\vec{P}) \vec{\sqsubseteq} \vec{\rho}_{ij}(\vec{P}) \vec{\sqsubseteq} \vec{P}$ follow from the observation that the composition $\vec{\rho}$ of reductions is a reduction and Lem. **14**. By def. of the quotient we have $\vec{P} \in [\vec{P}]/_{\vec{\equiv}}$ so, by def. of the reduced product $\vec{\gamma}([\vec{P}]/_{\vec{\equiv}}) = \vec{\gamma}(\vec{P}) = \vec{\gamma}(\vec{\rho}(\vec{P}))$ since $\vec{\rho}$ is sound proving the reduced product to be sound. It follows, by transfinite induction on the iterates, that $\forall \lambda : \vec{\rho}^{\lambda}(\vec{P}) = \vec{\gamma}([\vec{P}]/_{\vec{\equiv}})$ and so, passing to the limit, that if γ preserves lower bounds, $\vec{\rho}^{\star}(\vec{P}) = \vec{\gamma}([\vec{P}]/_{\vec{\equiv}})$ proving $\vec{\rho}^{\star}(\vec{P}), \vec{\rho}^{\lambda}(\vec{P}), \vec{\rho}_{ij}(\vec{P}), \vec{P} \in [\vec{P}]/_{\vec{\equiv}}$ by def. of equivalence classes of $\vec{\equiv}$ as well as soundness since $\vec{\rho}$ is sound. □

The following theorem proves that the iterated reduction may not be as precise as the reduced product. It is nevertheless easier to implement.

Theorem 16. *In general $\vec{\rho}^{\star}(\vec{P})$ may not be a minimal element of the reduced product class $[\vec{P}]/_{\vec{\equiv}}$ (in which case $\exists \vec{Q} \in [\vec{P}]/_{\vec{\equiv}} : \vec{Q} \vec{\sqsubset} \vec{\rho}^{\star}(\vec{P})$).*

Proof. Let $L = \wp(\{a, b, c\})$, $A_1 = \{\emptyset, \{a\}, \top\}$, $A_2 = \{\emptyset, \{a, b\}, \top\}$ et $A_3 = \{\emptyset, \{a, c\}, \top\}$ where $\top = \{a, b, c\}$. We have $\langle \top,\ \{a, b\},\ \{a, c\}\rangle/_{\vec{\equiv}} = \langle \{a\},\ \{a, b\},\ \{a, c\}\rangle$. However $\vec{\rho}^{*}_{\vec{ij}}(\langle \top,\ \{a, b\},\ \{a, c\}\rangle) = \langle \top,\ \{a, b\},\ \{a, c\}\rangle$ for $\Delta = \{1, 2, 3\}$, $i, j \in \Delta, i \neq j$ and so $\vec{\rho}^{*}(\langle \top,\ \{a, b\},\ \{a, c\}\rangle) = \langle \top,\ \{a, b\},\ \{a, c\}\rangle$ proving, for that example, that $\vec{\rho}^{\star}(\vec{P})$ is not a minimal element of $[\vec{P}]/_{\vec{\equiv}}$. □

Sufficient conditions exist for the iterated pairwise reduction to be a total reduction to the reduced product.

Theorem 17. *If the $\langle A_i, \sqsubseteq_i, \sqcup_i\rangle$, $i \in \Delta$ are complete lattices, the ρ_{ij}, $i, j \in \Delta$, $i \neq j$, are lower closure operators, and $\forall \vec{P}, \vec{Q} : \left(\vec{\gamma}\left(\vec{P}\right) \subseteq \vec{\gamma}\left(\vec{Q}\right)\right) \Rightarrow \left(\exists n \geqslant 0 : \left(\dot{\vec{\prod}}_{\substack{i,j \in \Delta, \\ i \neq j}} \vec{\rho}_{ij}\right)^{n}(\vec{P}) \vec{\sqsubseteq} \vec{Q}\right)$ then $\forall \vec{P} : \vec{\rho}^{\star}(\vec{P})$ is the minimum of the class $\vec{P}/_{\vec{\equiv}}$.*

Proof. First, a direct consequence of Th. **15** is that for all n, $\vec{\rho}^{\star} \dot{\vec{\sqsubseteq}} \left(\dot{\vec{\prod}}_{\substack{i,j \in \Delta, \\ i \neq j}} \vec{\rho}_{ij}\right)^{n}$. Let $\vec{Q} \in \vec{P}/_{\vec{\equiv}}$. Then $\vec{\gamma}\left(\vec{P}\right) = \vec{\gamma}\left(\vec{Q}\right)$, so $\vec{\gamma}\left(\vec{P}\right) \subseteq \vec{\gamma}\left(\vec{Q}\right)$. By hypothesis, that implies there is an n such that $\left(\dot{\vec{\prod}}_{\substack{i,j \in \Delta, \\ i \neq j}} \vec{\rho}_{ij}\right)^{n}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$, and so $\vec{\rho}^{\star}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$. □

5.3 (Reduced) product transformers

The transformers $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]$, $\mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]$, and $\mathsf{p}_{\mathcal{I}}[\![\varphi]\!]$ for the (pairwise iterated) reduced product proceed componentwise and reduce the result. This can be improved in the abstract, as follows.

Let us consider a reduced product $\langle (\bigtimes_{i \in \Delta} A_i)/_{\vec{\equiv}}, \vec{\sqsubseteq}\rangle$ of abstract domains $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$ with concretizations $\gamma_i \in A_i \xrightarrow{\nearrow} C$ and sound transformers $\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!]$ such that $\mathsf{f}[\![\mathtt{x} := t]\!]\gamma_i(\overline{P}) \subseteq \gamma_i(\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!]\overline{P})$ where $\mathsf{f}[\![\mathtt{x} := t]\!] \in C \xrightarrow{\nearrow} C$ is the increasing concrete transformer. The corresponding transformer of a property $\vec{P} \in \bigtimes_{i \in \Delta} A_i$ in the product is the reduction $\left(\bigtimes_{i \in \Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right)/_{\vec{\equiv}}$ of the componentwise transformation. This is sound since

$$
\begin{aligned}
& \vec{\gamma}\left(\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right)_{/\equiv}\right) \\
=\ & \vec{\gamma}\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right) && \text{〈def. reduced product〉} \\
=\ & \bigcap_{i\in\Delta} \gamma_i(\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)) && \text{〈def. } \vec{\gamma}\text{〉} \\
\supseteq\ & \bigcap_{i\in\Delta} \mathsf{f}[\![\mathtt{x} := t]\!](\gamma_i(\vec{P}_i)) && \text{〈soundness of the } \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!]\text{〉} \\
\supseteq\ & \mathsf{f}[\![\mathtt{x} := t]\!]\left(\bigcap_{i\in\Delta} \gamma_i(\vec{P}_i)\right) && \text{〈}\mathsf{f}[\![\mathtt{x} := t]\!] \text{ increasing〉} \\
=\ & \mathsf{f}[\![\mathtt{x} := t]\!](\vec{\gamma}(\vec{P})) && \text{〈def. } \vec{\gamma}\ .\text{〉}
\end{aligned}
$$

However this definition of the product transformer is not modular since it must be entirely redesigned when adding a new abstract domain to the product. Notice however, that abstract transformers themselves are elements of a reduced product, by defining their concretization as

$$
\begin{aligned}
& \vec{\gamma}\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right) \\
=\ & \bigcap_{i\in\Delta} \gamma_i(\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)) && \text{〈def. } \vec{\gamma}\text{〉} \\
=\ & \bigcap_{i\in\Delta} \dot{\gamma}_i(\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!])(\vec{P}) && \text{〈pointwise definition } \dot{\gamma}_i(f)(\vec{x}) \triangleq \gamma_i(f(\vec{x_i}))\text{〉} \\
=\ & \left(\dot{\bigcap_{i\in\Delta}} \dot{\gamma}_i(\bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!])\right)(\vec{P}) && \text{〈pointwise def. } \left(\dot{\bigcap_{i\in\Delta}} f_i\right)(x) \triangleq \bigcap_{i\in\Delta} f_i(x)\text{〉} \\
=\ & \vec{\gamma}\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!]\right)(\vec{P}) && \text{〈def. } \vec{\gamma}(\textstyle\bigtimes_i x_i) \triangleq \bigtimes_i \gamma_i(x_i)\text{) for products.〉}
\end{aligned}
$$

A direct consequence is that we can approximate the product transformer by iterated reduction of the componentwise transformers. Observe that we have the following Galois connection

$$
\begin{aligned}
& \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \subseteq Q \\
\Leftrightarrow\ & \{\langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \mid I \in \mathcal{I} \wedge \langle I,\ \eta\rangle \in P)\} \subseteq Q && \text{〈def. } \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]\text{〉} \\
\Leftrightarrow\ & \forall I \in \mathcal{I} : \forall \langle I,\ \eta\rangle \in P : \langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \in Q && \text{〈def. } \subseteq\text{〉} \\
\Leftrightarrow\ & \forall \langle I,\ \eta\rangle \in P : I \in \mathcal{I} \wedge \langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \in Q \\
& && \text{〈since } \langle I,\ \eta\rangle \in P \text{ implies } I \in \mathcal{I}\text{〉} \\
\Leftrightarrow\ & P \subseteq \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \in Q \right\} && \text{〈def. } \subseteq\text{〉} \\
& P \subseteq \mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]Q && \text{〈def. } \mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]\ .\text{〉}
\end{aligned}
$$

It follows that if $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \subseteq Q$ then $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!](\mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!](Q))$ is a more precise sound overapproximation of $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P$ than Q, which suggests the following

pairwise reduction $\dot{\rho}_{ij}$ of transformers (based on the pairwise reduction ρ_{ij} of abstract properties)

$$\begin{array}{l}\dot{\rho}_{ij}(\langle \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!],\ \bar{\mathsf{f}}_j[\![\mathtt{x} := t]\!]\rangle) \quad \triangleq \\ \quad \boldsymbol{\lambda}\,\langle x,\ y\rangle \bullet \text{let } \langle x',\ y'\rangle \triangleq \rho_{ij}(\langle \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](x),\ \bar{\mathsf{f}}_j[\![\mathtt{x} := t]\!](y)\rangle) \text{ in} \\ \qquad \text{let } \langle `x,\ `y\rangle \triangleq \rho_{ij}(\langle x \sqcap_i \overline{\mathsf{b}}_i[\![\mathtt{x} := t]\!](x),\ y \sqcap_j \overline{\mathsf{b}}_j[\![\mathtt{x} := t]\!](y)\rangle) \text{ in} \\ \qquad\quad \rho_{ij}(\langle \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](`x),\ \bar{\mathsf{f}}_j[\![\mathtt{x} := t]\!](`y)\rangle)\end{array}$$

which defines a reduction $\dot{\vec{\rho}}$ of transformers by (1) lifting the reduction $\vec{\rho}$ on the product of abstract properties at higher-order.

Example 18. Consider the product of equality and sign analysis. The componentwise forward propagation of $\langle \mathtt{a} = \mathtt{b},\ \top\rangle$ through the assignment $\mathtt{a} := \sqrt{\mathtt{b}} + \mathtt{a}$ is $\langle \top,\ \mathtt{b} \geqslant 0\rangle$ (with runtime error when $\mathtt{b} < 0$ in which case execution is assumed to stop). The backward propagation yields the precondition $\langle \mathtt{a} = \mathtt{b},\ \mathtt{b} \geqslant 0\rangle$ reduced to $\langle \mathtt{a} = \mathtt{b},\ \mathtt{b} \geqslant 0 \wedge \mathtt{a} \geqslant 0\rangle$ which forward propagation is now reduced to $\langle \top,\ \mathtt{a} \geqslant 0 \wedge \mathtt{b} \geqslant 0\rangle$. So the reduced componentwise forward propagation of $\langle \mathtt{a} = \mathtt{b},\ \top\rangle$ through the assignment $\mathtt{a} := \sqrt{\mathtt{b}} + \mathtt{a}$ is $\langle \top,\ \mathtt{a} \geqslant 0 \wedge \mathtt{b} \geqslant 0\rangle$ (more precise than $\langle \top,\ \mathtt{b} \geqslant 0\rangle$). □

Similar reductions can be done for the backward assignment $\mathsf{b}[\![\mathtt{x} := t]\!]$ (thus generalizing [15]) and tests $\mathsf{p}[\![\varphi]\!]$.

Example 19. An iterated reduction of the product of linear equalities and sign analyses of $\mathsf{p}[\![(\mathtt{x} = \mathtt{y}) \wedge ((\mathtt{z} + 1) = \mathtt{x}) \wedge (\mathtt{y} = \mathtt{z})]\!]$ with precondition $\mathtt{x} = 0$ yields the postcondition $\mathtt{x} = 0 \wedge \mathtt{y} = 0 \wedge \mathtt{z} < 0$, see the details in [5, Sect. 13.9]. □

5.4 Widening

The widening/narrowing [6] of a reduced product is often defined componentwise using widenings/narrowings of the component abstract domains. This ensures convergence for the product. However it must be proved that the reduction does not break down the termination of the product widening, in which case reduction must be weakened or the widening strengthened.

Example 20. The closure operation in the octagon abstract domain can be considered as a reduction between separate domains, each considering only a pair of variables: if one applies the classical widening operation on octagons followed by closure (reduction), then termination is no longer ensured (e.g. see [22, Fig. 25–26]). □

5.5 Observational reduced product

The observational reduced product of abstract domains $\langle A_i,\ \sqsubseteq_i\rangle$, $i \in \Delta$ consists in introducing observables to increase the precision of the Cartesian product. We will write ${}^{\Omega}\bigtimes_{i\in\Delta} A_i$ for the *observational Cartesian product* with observables named by Ω. It can be seen as the application of the extension operator of Sect. **3** followed by a Cartesian product $\bigtimes_{i\in\Delta} A_i$. This operation is not very fruitful, as the shared observables will not bring much information. But used in conjunction with an iterated reduction, it can give very precise results since information about the observables can bring additional reductions.

Definition 21 (Observational reduced product). *For all $i \in \Delta$, let $\langle {}^iA_{\mathcal{I}}^{\Sigma_O}, {}^i\sqsubseteq\rangle$, $\langle {}^iA_{\mathcal{I}}^{\Sigma_{O'}}, {}^i\sqsubseteq'\rangle$ be abstract domains, Ω' be the new observables, and ${}^i\mathsf{extend}_{\Omega'} \in {}^iA_{\mathcal{I}}^{\Sigma_O} \to {}^iA_{\mathcal{I}}^{\Sigma_{O'}}$ be the sound extensions satisfying the conditions of definition **12**.*

The observational cartesian product *is*

$$ {}^{\Omega'}\bigtimes_{i\in\Delta} {}^iA_{\mathcal{I}}^{\Sigma_O} \triangleq \bigtimes_{i\in\Delta} {}^i\mathsf{extend}_{\Omega'}\left[{}^iA_{\mathcal{I}}^{\Sigma_O}\right] $$

and the observational reduced product *is* $\langle \left({}^{\Omega}\bigtimes_{i\in\Delta} A_i\right)/_{\equiv}, \vec{\sqsubseteq}\rangle$.

6 The Nelson-Oppen combination procedure

6.1 Formula purification

6.1.1 Formula purification in the Nelson-Oppen theory combination procedure

Given disjoint deductive theories $\mathcal{T}_i$ in $\mathbb{F}(\Sigma_i)$, $\Sigma_i \dot\subseteq \Sigma$ with equality and decision procedures sat_i for satisfiability of quantifier-free conjunctive formulæ $\varphi_i \in \mathbb{C}(\Sigma_i)$, $i = 1, ..., n$, the Nelson-Oppen combination procedure [23] decides the satisfiability of a quantifier-free conjunctive formula $\varphi \in \mathbb{C}(\bigcup_{i=1}^n \Sigma_i)$ in theory $\mathcal{T} = \bigcup_{i=1}^n \mathcal{T}_i$ such that $\mathfrak{M}(\mathcal{T}) = \bigcap_{i=1}^n \mathfrak{M}(\mathcal{T}_i)$.

The first "purification" phase [29, Sect. 2] of Nelson-Oppen combination procedure consists in repeating the replacement of (all occurrences of) an alien subterm $t \in \mathbb{T}(\Sigma_i)\setminus \mathbb{x}$ of a subformula $\psi[t] \notin \mathbb{C}(\Sigma_i)$ (including equality or inequality predicates $\psi[t] = (t = t')$ or $(t' = t)$) of φ by a fresh variable $x \in \mathbb{x}$ such that $\#(\mathbb{x}) = \#(t)$ and introducing the equation $x = t$ (i.e. $\varphi[\psi[t]]$ is replaced by $\varphi[\psi[x]] \wedge x = t$ and the replacement is recursively applied to $\varphi[\psi[x]]$ and $x = t$).

Example 22 (Formula purification). Assume $f \in \mathbb{f}_1$ and $g \in \mathbb{f}_2$. $\varphi = (g(\mathbf{x}) = f(g(g(\mathbf{x})))) \to (\exists y : y = f(g(y)) \wedge y = g(\mathbf{x})) \to (\exists y : \exists z : y = f(z) \wedge y = g(\mathbf{x}) \wedge z = g(y)) \to (\exists y : \exists z : \varphi_1 \wedge \varphi_2) = \varphi'$ where $\varphi_1 = (y = f(z))$ and $\varphi_2 = (y = g(\mathbf{x}) \wedge z = g(y))$. □

Upon termination, the quantifier-free conjunctive formula φ is transformed into a formula φ' of the form

$$ \varphi' \quad = \quad \exists \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n} \varphi_i \quad \text{where} \quad \varphi_i \quad = \quad \varphi_i' \wedge \bigwedge_{x_i \in \vec{x}_i} x_i = t_{x_i}, $$

$\vec{x} \triangleq \bigcup_{i=1}^n \vec{x}_i$ is the set of auxiliary variables introduced by the purification , each $t_{x_i} \in \mathbb{T}(\Sigma_i)$ is an alien subterm of φ renamed as $x_i \in \mathbb{x}$ such that $\#(x_i) = \#(t_{x_i})$, and each φ_i' (hence each φ_i) is a quantifier-free conjunctive formula in $\mathbb{C}(\Sigma_O^i)$. We have $\varphi \Leftrightarrow \bigwedge_{i=1}^n \varphi_i'[x_i \leftarrow t_{x_i}]_{x_i \in \vec{x}_i}$ so φ and φ' are equisatisfiable.

In case of non-disjoint theories $\mathcal{T}_i$, $i = 1, ..., n$, purification is still possible, by considering the worst case (so as to purify any subterm of theories $\mathcal{T}_i$ or $\mathcal{T}_j$ occurring in a term of theories $\mathcal{T}_i$ or $\mathcal{T}_j$). The reason the Nelson-Oppen purification requires disjointness of theory signatures is that otherwise they can share more than equalities and cardinality, a sufficient reason for the procedure to be incomplete. Nevertheless, the purification procedure remains sound for non-disjoint theories, which can be exploited for static analysis, as shown below.

6.1.2 The Nelson-Oppen purification as an observational cartesian product

Let the observable identifiers be the free variables of $\varphi \in \mathbb{C}(\Sigma)$, $\mathbb{x}_\mathrm{P} = \vec{\mathbb{x}}_\varphi$ plus the fresh auxiliary variables $\vec{x}$ introduced by the purification $\mathbb{x}_O = \mathbb{x}_\mathrm{P} \cup \vec{x}$. Let Σ_P and Σ_O be the corresponding signatures of Σ. Given an interpretation $I \in \mathcal{I}$, with values $I_\mathcal{V}$, the observable naming Ω^φ_I is

$$\begin{array}{rcll} \Omega^\varphi_I & \in & \mathbb{x}_O \to \mathcal{R}^{\Sigma_\mathrm{P}}_I \to I_\mathcal{V} & \\ \Omega^\varphi_I(x)\eta & \triangleq & \eta(x) & \text{when} \quad x \in \mathbb{x}_\mathrm{P}, \\ & \triangleq & [\![t_x]\!]\eta & \text{when} \quad x \in \vec{x} \quad . \end{array}$$

From a model-theoretic point of view, the purification of $\varphi \in A$ into $\langle \varphi_1, \ldots, \varphi_n \rangle$ can be considered as an abstraction of the program properties in $\mathfrak{P}^{\Sigma_O}_\mathcal{I}$ abstracted by φ to observable properties in $\mathfrak{R}^{\Sigma_O}_\mathcal{I}$ themselves abstracted to the observational cartesian product ${}^{\Omega^\varphi}\!\bigtimes_{i\in\Delta} {}^i A^{\Sigma_O}_\mathcal{I}$ where the component abstract domains are $\langle {}^i A^{\Sigma_O}_\mathcal{I}, \sqsubseteq_i \rangle \triangleq \langle \mathbb{C}(\Sigma^i_O), \Rightarrow \rangle$ with concretizations ${}^i\gamma^{\Sigma_O}_\mathcal{I} \in \mathbb{C}(\Sigma^i_O) \to {}^i\mathfrak{P}^{\Sigma_O}_\mathcal{I}$ such that ${}^i\gamma^{\Sigma_O}_\mathcal{I} \in \mathbb{C}(\Sigma^i_O) \to {}^i\mathfrak{P}^{\Sigma_O}_\mathcal{I}$ and ${}^i\gamma^{\Sigma_O}_\mathcal{I}(\varphi) \triangleq \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_O}_\mathcal{I} \;\middle|\; I \in \mathfrak{M}(\mathcal{T}_i) \wedge I \models_\eta \varphi \right\}$, $i = 1, \ldots, n$. This follows from the fact that the concretization is the same, as shown below.

$$\begin{array}{l} \boldsymbol{\gamma}^{\Omega^\varphi,\mathrm{P}}_\mathcal{I}\left({}^{\Omega^\varphi}\!\bigtimes_{i=1}^{n} \varphi'_i \right) \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta \rangle \in \gamma^{\Sigma_O}_\mathcal{I}\left({}^{\Omega^\varphi}\!\bigtimes_{i=1}^{n} \varphi'_i \right) \right\} \\ \qquad\qquad \wr \text{def. } \boldsymbol{\gamma}^{\Omega^\varphi,\mathrm{P}}_\mathcal{I} \triangleq \boldsymbol{\gamma}^{\Omega^\varphi}_\mathcal{I} \circ \gamma^{\Sigma_O}_\mathcal{I} \wr \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega^\varphi_x(\eta) \rangle \in \bigcap_{i=1}^{n} {}^i\gamma^{\Sigma_O}_\mathcal{I}(\varphi'_i) \right\} \\ \qquad\qquad \wr \text{def. } \gamma^{\Sigma_O}_\mathcal{I} \text{ for the observational cartesian product} \wr \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega^\varphi_x(\eta) \rangle \in \gamma^{\Sigma_O}_\mathcal{I}\left(\bigwedge_{i=1}^{n} \varphi'_i \right) \right\} \\ \qquad\qquad \wr \text{def. } \gamma^{\Sigma_O}_\mathcal{I}(\Psi) \triangleq \left\{ \langle I, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge I \models_\eta \Psi \right\} \text{ and } \models \wr \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \in \mathbb{x}_\mathrm{P} \bullet \eta(x) \,\dot\cup\, \boldsymbol{\lambda}\, x \in \vec{x} \bullet [\![t_x]\!]\eta \rangle \in \gamma^{\Sigma_O}_\mathcal{I}\left(\bigwedge_{i=1}^{n} \varphi'_i \right) \right\} \\ \qquad\qquad \wr \text{def. } \Omega^\varphi_x, \mathbb{x} = \mathbb{x}_\mathrm{P} \cup \vec{x}, \text{ and } \mathbb{x}_\mathrm{P} \cap \vec{x} = \emptyset \wr \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \eta \rangle \in \gamma^{\Sigma_O}_\mathcal{I}\left(\exists\, \vec{x} : \bigwedge_{i=1}^{n} \varphi'_i \wedge \bigwedge_{x\in\vec{x}} x = t_x \right) \right\} \\ \qquad\qquad \wr \text{def. } \gamma^{\Sigma_O}_\mathcal{I} \text{ and } \models \wr \\ = \left\{ \langle I, \eta \rangle \in \mathfrak{R}^{\Sigma_\mathrm{P}}_\mathcal{I} \;\middle|\; \langle I, \eta \rangle \in \gamma^{\mathrm{P}}_\mathcal{I}\left(\exists\, \vec{x} : \bigwedge_{i=1}^{n} \varphi'_i \wedge \bigwedge_{x\in\vec{x}} x = t_x \right) \right\} \end{array}$$

$$= \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}_{\mathcal{I}}^{\Sigma_{\mathrm{P}}} \;\middle|\; \langle I,\ \eta\rangle \in \gamma_{\mathcal{I}}^{\mathrm{P}}\left(\exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n}\left(\varphi_i' \wedge \bigwedge_{x_i \in \vec{x}_i} x_i = t_{x_i}\right)\right) \right\}$$

$$= \gamma_{\mathcal{I}}^{\mathrm{P}}\left(\exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n}\left(\varphi_i' \wedge \bigwedge_{x_i \in \vec{x}_i} x_i = t_{x_i}\right)\right) \qquad \text{⟅def. } \{\, x \mid P(x)\,\}\text{⟆}$$

$$= \gamma_{\mathcal{I}}^{\mathrm{P}}\left(\exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n} \varphi_i\right) \qquad \text{⟅def. } \varphi_i = \varphi_i' \wedge \bigwedge_{x_i \in \vec{x}_i} x_i = t_{x_i}\text{⟆}$$

$$= \gamma_{\mathcal{I}}^{\mathrm{P}}(\varphi') \qquad \text{⟅def. } \varphi' = \exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n} \varphi_i\ .\text{⟆}$$

After purification, the components of the observational cartesian product are not yet the most precise ones.

6.2 Formula reduction

6.2.1 Formula reduction in the Nelson-Oppen theory combination procedure

After purification, the Nelson-Oppen combination procedure [23] includes a reduction phase where all variable equalities $x = y$ and inequalities $x \neq y$ information available from one component φ_i in its theory $\mathcal{T}_i$ are propagated to all components φ_j (in practice only to those components φ_j where the information is useful, that is those φ_j, including φ_i, sharing free variables x and y with φ_i). The decision procedure for $\mathcal{T}_i$ is used to determine all possible disjunctions of conjunctions of (in)equalities that are implied by φ_i. These are determined by exhaustively trying all possibilities in the nondeterministic version of the procedure or by an incremental construction in the deterministic version, which is more efficient for convex theories [29]. The reduction is iterated until no new disjunction of (in)equalities is found.

6.2.2 The Nelson-Oppen reduction as an iterated fixpoint reduction of the product

Let $\mathbb{1}_s \triangleq \{\, \langle s,\ s\rangle \;\big|\; s \in S \,\}$ be the identity relation on a set S and $\mathfrak{E}(S)$ be the set of all equivalence relations on S that is $\mathfrak{E}(S) \triangleq \left\{ r \in \wp(S \times S) \;\middle|\; \mathbb{1}_s \subseteq r \wedge r = r^{-1} \wedge r = r \circ r \right\}$. Define the pairwise reduction

$$\begin{array}{rcl} \rho_{ij}(\varphi_i, \varphi_j) & \triangleq & \langle \varphi_i \wedge E_{ij} \wedge E_{ji},\ \varphi_j \wedge E_{ji} \wedge E_{ij}\rangle \quad \text{where} \\ \mathrm{eq}(E) & \triangleq & \displaystyle\bigvee_{\equiv \in E}\left(\bigwedge_{x \equiv y} x = y \wedge \bigwedge_{x \not\equiv y} x \neq y\right) \\ E_{ij} & \triangleq & \displaystyle\bigwedge\left\{ \mathrm{eq}(E) \;\middle|\; E \subseteq \mathfrak{E}(\vec{\mathbf{x}}_{\varphi_i} \cap \vec{\mathbf{x}}_{\varphi_j}) \wedge \varphi_i \Rightarrow \mathrm{eq}(E) \right\}\ . \end{array}$$

The Nelson-Oppen reduction of φ purified into ${}^{\Omega^\varphi}\bigtimes_{i=1}^{n}\varphi'_i$ consists in computing the iterated pairwise reduction $\vec{\rho}^{\,*}\left({}^{\Omega^\varphi}\bigtimes_{i=1}^{n}\varphi'_i\right)$.

Example 23. Let $\varphi_1 \triangleq (x = \mathtt{a} \vee x = \mathtt{b}) \wedge y = \mathtt{a} \wedge z = \mathtt{b}$ and $\varphi_2 \triangleq \mathtt{f}(x) \neq \mathtt{f}(y) \wedge \mathtt{f}(x) \neq \mathtt{f}(z)$ so that $\varphi \triangleq \varphi_1 \wedge \varphi_2$ is purified. We have $E_{12} \triangleq (x = y) \vee (x = z)$ and $E_{21} \triangleq (x \neq y) \wedge (x \neq z)$ so that $\vec{\rho}^{\,*}(\varphi) = \mathtt{ff}$. □

Observe that the result of the iterated pairwise reduction may not be as precise as the reduced product.

Example 24. A classical example is given by [29, p. 11] where $\varphi_1 \triangleq \mathtt{f}(x) \neq \mathtt{f}(y)$ in the theory of Booleans admitting models of cardinality at most 2 and $\varphi_2 \triangleq \mathtt{g}(x) \neq \mathtt{g}(z) \wedge \mathtt{g}(y) \neq \mathtt{g}(z)$ in a disjoint theory admitting models of any cardinality so that $\varphi = \varphi_1 \wedge \varphi_2$ is purified. The reduction yields $\varphi \wedge x \neq y \wedge x \neq z \wedge y \wedge z$ and not $\mathtt{ff}$ since the cardinality information is not propagated whereas it would be propagated by the reduced product which is defined at the interpretation level. Therefore the pairwise reduction ought to be refined to include cardinality information, as proposed by [32]. □

6.2.3 Formula reduction and the reduced product

A formula over a set of theories is equivalent to its purification, so that to find an invariant or to check that a formula is invariant, we could first purify it and then proceed with the computation of the transformer of the program. This would lead to the same result as simply using one mixed formula if the reduction is total at each step of the computation. Such a process would be unnecessarily expensive if decision procedures could handle arbitrary formulæ. But this is not the case actually: most of the time, they cannot deal with quantifiers, and assignments introduce existential quantifiers which have to be approximated. Such approximations have to be redesigned for each set of formulæ. Using a reduced product of formulæ on base theories allows reusing the approximations on each theory (as in [19], even if the authors didn't recognize the reduced product). In that way, a reduced product of logical abstract domains will provide a modular approach to invariant proofs.

6.3 Formula satisfiability

After purification and reduction, the Nelson-Oppen combination procedure [23] includes a decision phase to decide satisfiability of the formula by testing the satisfiability of its purified components. This phase can also be performed during the program static analysis since an unsatisfiability result means unreachability encoded by $\mathtt{ff}$. The satisfiability decision can also be used as an approximation to check for a postfixpoint and that the specification is satisfied.

For briefness, we have concentrated in this paper on the Nelson-Oppen combination procedure [23] but Shostak combination procedure [27, 26] can be handled in exactly the same way.

7 Reduced product of logical and algebraic abstract domains

7.1 Combining logical and algebraic abstract domains

Static analyzers such as Astrée [1, 10] and Clousot [14] are based on an iterated pairwise reduction of a product of abstract domains over-approximating their reduced product [11]. Since logical abstract domains as combined by Nelson-Oppen combination procedure are indeed an iterated pairwise reduction of a product of abstract domains over-approximating their reduced product, as shown in Sect. **6.2**, the design of abstract interpreters based on an approximation of the reduced product can use both logical and algebraic abstract domains.

An advantage of using a product of abstract domains with iterated reductions is that the reduction mechanism can be implemented once for all while the addition of a new abstract domain to improve precision essentially requires the addition of a reduction with the other existing abstract domains when necessary [11].

Notice that the Nelson-Oppen procedure and its followers aim at completeness, at least for the reduction to $\mathtt{ff}$. However this is not needed in the iterated pairwise reduction used in static analysis since the reduction generally needs not to be optimal, e.g. for performance motivations, so that, in that context, restrictions to e.g. stably-infinite theories [29] or other similar hypotheses on interpretations [32] become irrelevant.

Similarly, the disjointness of the signatures of theories is essentially to achieve completeness. Instead, one may assume that the theories are consistent i.e. agree on the interpretations of their common signature, up to an isomorphism [30].

Example 25. As a simple example, consider the combination of the logical domain of Presburger arithmetics (where the multiplication is inexpressible) and the algebraic domain of sign analysis (which is complete for multiplication). The abstraction of a first-order formula to a formula of Presburger arithmetics is by abstraction to a subsignature eliminating all terms of the signature not in the subsignature:

$$
\begin{array}{rcl}
\alpha_\Sigma(\mathtt{x}) & \triangleq & \mathtt{x} \\
\alpha_\Sigma(\mathtt{f}(t_1,\ldots,t_n)) & \triangleq & ?, \quad \mathtt{f} \notin \Sigma \vee \exists\, i \in [1,n] : \alpha_\Sigma(t_i) = ? \\
& \triangleq & \mathtt{f}(t_1,\ldots,t_n), \quad \text{otherwise} \\
\alpha_\Sigma(\mathtt{ff}) & \triangleq & \mathtt{ff} \\
\alpha_\Sigma(\mathtt{p}(t_1,\ldots,t_n)) & \triangleq & \mathtt{tt}, \quad \mathtt{p} \notin \Sigma \vee \exists\, i \in [1,n] : \alpha_\Sigma(t_i) = ?, \text{ in positive position} \\
& \triangleq & \mathtt{ff}, \quad \mathtt{p} \notin \Sigma \vee \exists\, i \in [1,n] : \alpha_\Sigma(t_i) = ?, \text{ in negative position} \\
& \triangleq & \mathtt{p}(t_1,\ldots,t_n), \quad \text{otherwise} \\
\alpha_\Sigma(\neg\Psi) & \triangleq & \neg\alpha_\Sigma(\Psi) \\
\alpha_\Sigma(\Psi \wedge \Psi') & \triangleq & \alpha_\Sigma(\Psi) \wedge \alpha_\Sigma(\Psi')) \\
\alpha_\Sigma(\exists\, \mathtt{x} : \Psi) & \triangleq & \exists\, \mathtt{x} : \alpha_\Sigma(\Psi)\ .
\end{array}
$$

The abstract transformers for Presburger arithmetics become simply $\mathtt{f}_{\mathcal{P}}[\![\mathtt{x} := e]\!]P \triangleq \alpha_{\Sigma_{\mathcal{P}}}(\exists\, x' : P[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = e[\mathtt{x} \leftarrow x'])$, $\mathtt{p}_{\mathcal{P}}[\![\varphi]\!]P \triangleq \alpha_{\Sigma_{\mathcal{P}}}(P \wedge \varphi)$, etc, where $\Sigma_{\mathcal{P}}$ is the signature of Presburger arithmetics.

The reduction of the Presburger arithmetics logical abstract domain by the sign algebraic abstract domain is given by the concretization function for signs.

$$E_{ij}(\overline{\eta}) \triangleq \bigwedge_{\mathtt{x}\in\mathsf{dom}(\overline{\eta})} \gamma(\mathtt{x}, \overline{\eta}(\mathtt{x})) \qquad \begin{array}{lll} \gamma(\mathtt{x}, \mathtt{pos0}) & \triangleq & (\mathtt{x} \geqslant 0) \\ \gamma(\mathtt{x}, \mathtt{pos}) & \triangleq & (\mathtt{x} > 0) \quad \text{etc.} \end{array}$$

Assume the precondition $\langle P(\mathtt{x}),\ \mathtt{x} : \top\rangle$ holds, then after the assignment $\mathtt{x} := \mathtt{x} \times \mathtt{x}$, the post condition $\langle \exists\, x' : P(x') \wedge \mathtt{x} = x' \times x',\ \mathtt{x} : \mathtt{pos0}\rangle$ holds, which must be abstracted by $\alpha_{\Sigma_{\mathcal{P}}}$ to the Presburger arithmetics logical abstract domain that is $\langle \exists\, x' : P(x'),\ \mathtt{x} : \mathtt{pos0}\rangle$. The reduction reduces the postcondition to $\langle \exists\, x' : P(x') \wedge x \geq 0,\ \mathtt{x} : \mathtt{pos0}\rangle$.

Symmetrically, the sign abstract domain may benefit from equality information. For example, if the sign of `x` is unknown then it would remain unknown after the code `y := x; x := x * y` whereas knowing that `x = y` is enough to conclude that `x` is positive.

Of course the same result could be achieved by encoding by hand the Presburger arithmetics transformer for the assignment to cope with this case and other similar ones. Here the same result is achieved by the reduction without specific programming effort for each possible particular case. □

7.2 Reduced product for inconsistent interpretations

One of the issues with the use of logical abstract domains, or even with the use of SMT solvers to prove invariants, is that the underlying theory is often not sound with respect to the actual implementations of the program.

Example 26. Astrée [1, 10] has found runtime error bugs in programs which had been "formally proved correct". The problem was a buffer overrun. Indeed the theory of arrays used in the "formal proof" was not taking buffer overruns into account [3]. In practice, this can have dramatic consequences, as shown by the following example

```
#include <stdio.h>
int main () { int n, T[1];
   n = 2147483647;
   printf("n = %i, T[n] = %i\n", n, T[n]);
}
```

producing quite different results on different machines:

`n = 2147483647, T[n] = 2147483647`	Macintosh PPC
`n = 2147483647, T[n] = -1208492044`	Macintosh Intel
`n = 2147483647, T[n] = -135294988`	PC Intel 32 bits
`Bus error`	PC Intel 64 bits

This example also shows that any attempt to define the machine semantics is hopeless. In practice, one can consider such cases as errors and stop the analysis when they are encoutered. It is also possible to keep the analysis running after reporting such cases, but the meaning of the analysis is not a sound approximation of all program behaviors

anymore, just an approximation of the execution traces that don't fall in that case. However, this unsoundness problem disappears when proving the absence of runtime error in such cases of unpredictable behavior after the error. □

However, recovering soundness is possible by introducing reduced products with well chosen abstract domains. For example, a logical abstract domain for mathematical integers can be combined with an algebraic abstract domain handling bounded machine integers. The coherence is achieved by a reduction of the logical abstract domain limiting the range of variation of program integer variables to that are discovered by the algebraic abstract domain.

Given two interpretations, I^1 and I^2 for a signature $\Sigma = \langle \mathbb{s}, \mathbb{x}, \mathbb{f}, \mathbb{p}, \# \rangle$, we define their common interpretation I such that ($\mathfrak{U}^s$ signals a runtime error when the two interpretations differ):

$$
\begin{array}{rcll}
I^s_{\mathcal{V}} & \triangleq & (I^{1,s}_{\mathcal{V}} \cap I^{2,s}_{\mathcal{V}}) \cup \{\mathfrak{U}^s\}^{\,4} & s \in \mathbb{s} \\
I_\gamma(\mathtt{c}) & \triangleq & I^1_\gamma(\mathtt{c}) & \text{when} \quad I^1_\gamma(\mathtt{c}) = I^2_\gamma(\mathtt{c}) \\
 & \triangleq & \mathfrak{U}^s & \text{otherwise } (\#(\mathtt{c}) = s) \\
I_\gamma(\mathtt{f}) & \triangleq & I^1_\gamma(\mathtt{f}) & \text{when } \#(\mathtt{f}) = s_1 \times \ldots \times s_n \to s \text{ and } \forall v_1 \in I^{s_1}_{\mathcal{V}}, \ldots, \forall v_n \in I^{s_n}_{\mathcal{V}} : I^1_\gamma(\mathtt{f})(v_1, \ldots, v_n) = I^2_\gamma(\mathtt{f})(v_1, \ldots, v_n) \\
 & \triangleq & \mathfrak{U}^s & \text{otherwise} \\
I_\gamma(\mathtt{p}) & \triangleq & I^1_\gamma(\mathtt{p}) & \text{when } \#(\mathtt{p}) = s_1 \times \ldots \times s_n \to \mathtt{bool} \text{ and } \forall v_1 \in I^{s_1}_{\mathcal{V}}, \ldots, \forall v_n \in I^{s_n}_{\mathcal{V}} : I^1_\gamma(\mathtt{p})(v_1, \ldots, v_n) = I^2_\gamma(\mathtt{p})(v_1, \ldots, v_n) \\
 & \triangleq & \mathfrak{U}^s & \text{otherwise}
\end{array}
$$

all logical operators $\neg$, $\wedge$, $\exists$ are strict in all errors $\mathfrak{U}^s$, $s \in \mathbb{s}$, the reduction is defined such that $\rho_{ij}(\langle \mathfrak{U}^s, Q \rangle) = \rho_{ij}(\langle P, \mathfrak{U}^s \rangle) = \rho_{ij}(\langle \mathfrak{U}^s, \mathfrak{U}^{s'} \rangle) = \mathfrak{U}^s$, and errors are interpreted as stopping program execution. It follows that the abstractions for different interpretations can be left unchanged since the reduction takes the errors cases into account during the static analysis.

The main consequence is that in absence of any error $\mathfrak{U}^s$, $s \in \mathbb{s}$, the iterated pairwise reduction of the two interpretations do coincide (more precisely up to the first error during execution, if any), so that we have a sound approximation of the actual program semantics.

7.3 Program purification

Whereas the reduced product proceeds componentwise, logical abstract domains often combine all these components into the single formula of their conjunction which is then globally propagated by property transformers before being purified again into components by the Nelson-Oppen procedure. These successive abstractions by purification and concretization by conjunction can be avoided when implementing the logical abstract domain as an iterated reduction of the product of the component and program purification, as defined below. The observational semantics is then naturally implemented by a program transformation.

[4] This condition could also be considered up to an isomorphism.

Given disjoint signatures $\langle \mathbb{f}_i,\ \mathbb{p}_i\rangle$, $i = 1, ..., n$, the purification of a program $\mathtt{P}$ over $\mathbb{C}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i, \bigcup_{i=1}^n \mathbb{p}_i)$ consists in purifying the terms t in its assignments $\mathtt{x} := e$ and the clauses in simple conjunctive normal form φ appearing in conditional or iteration tests. A term $t \in \mathbb{T}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i)$ not reduced to a variable is said "*to have type i*" when it is of the form $\mathtt{c} \in \mathbb{f}_i^0$ or $\mathtt{f}(t_1, \ldots, t_n)$ with $\mathtt{f} \in \mathbb{f}_i^n$. As a side note, one may observe that this could very well be equivalent to using the variable and term types in a typed language.

The purification of an assignment $\mathtt{x} := e[t]$ where term e has type i and the alien subterm t has type j, $j \neq i$ consists in replacing this assignment by $x = t; \mathtt{x} := e[x]$ where $x \in \mathbb{x}$ is a fresh variable of sort $\#(x) = \#(t)$, $e[x]$ is obtained from $e[t]$ by replacing all occurrences of the alien subterm t by the fresh variable x in e, and in recursively applying the replacement to $x = t$ and $\mathtt{x} := e[x]$ until no alien subterm is left.

An atomic formula $a \in \mathbb{A}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i, \bigcup_{i=1}^n \mathbb{p}_i)$ not reduced to $\mathtt{false}$ is said *to have type i* when it is of the form $\mathtt{p}(t_1, \ldots, t_n)$ with $\mathtt{p} \in \mathbb{p}_i^n$ or $t_1 = t_2$ and t_1 has type i or $\mathtt{x} = t_2$ and t_2 has type i.

The purification of an assignment $\mathtt{x} := a[t]$ where atomic formula a has type i and the alien subterm t has type j, $j \neq i$ consists in replacing this assignment by $x = t; \mathtt{x} := a[x]$ where $x \in \mathbb{x}$ is a fresh variable of sort $\#(x) = \#(t')$, $a[x]$ is obtained from $a[t]$ by replacing all occurrences of the alien subterm t by the fresh variable x, and in recursively applying the replacement to $x = t$ and $\mathtt{x} := a[x]$ until no alien subterm is left.

The purification of a clause in simple conjunctive normal form $\varphi \in \mathbb{C}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i, \bigcup_{i=1}^n \mathbb{p}_i)$ in a test consists in replacing all atomic subformulæ a (including equalities and disequalities) by fresh variables, in introducing assignments $x := a$ to these fresh variables x of sort $\#(x) = \mathtt{bool}$ before the test and in recursively purifying the assignments $x := a$.

Example 27. Assume that $f \in \mathbb{f}_1$ and $g \in \mathbb{f}_2$. The purification is

$\mathtt{if}\ (g(\mathtt{w}) = f(g(g(\mathtt{w})))\ \mathtt{then} \ldots$

$\rightarrow \quad x := (g(\mathtt{w}) = f(g(g(\mathtt{w})))); \mathtt{if}\ x\ \mathtt{then} \ldots$

$\rightarrow \quad y := g(\mathtt{w}); x := (y = f(g(y))); \mathtt{if}\ x\ \mathtt{then} \ldots$

$\wr g(\mathtt{w})$ has type 2 and $f(g(g(\mathtt{w})))$ has type $1\int$

$\rightarrow \quad y := g(\mathtt{w}); z := g(y); x := (y = f(z)); \mathtt{if}\ x\ \mathtt{then} \ldots$

$\wr (y = f(g(y)))$ has type 1 and $g(y)$ has type $2\ .\int$ □

After purification all program terms, atomic formulæ, and clauses are pure in that no term or atomic formula of a theory has a subterm in a different theory or a clause containing terms of different theories. So all term assignments $x := e$ (or atomic formulæ $x := a$) have $t \in \mathbb{T}(\Sigma_O^i)$ (resp. $a \in \mathbb{A}(\Sigma_O^i, \mathbb{p}_i)$ for some $i \in [1, n]$ and all clauses in tests are Boolean expressions written using only variables, $\neg$ and $\wedge$.

We let the observable identifiers $\mathbb{x}_O = \mathbb{x}_\mathrm{P} \cup \vec{x}$ be the program variables $\mathbb{x}_\mathrm{P}$ plus the fresh auxiliary variables $x \in \vec{x}$ introduced by the purification with assignments $x := e_x$. Given an interpretation I, with values $I_\mathcal{V}$, the observable naming Ω_I is

$$\begin{array}{rcll} \Omega & \in & \mathbb{x}_O \mapsto (\mathbb{x}_\mathrm{P} \mapsto I_\mathcal{V}) \mapsto I_\mathcal{V} & \\ \Omega_I(x)\eta & \triangleq & \eta(x) & \text{when} \quad x \in \mathbb{x}_\mathrm{P} \\ & \triangleq & [\![e_x]\!]\eta & \text{when} \quad x \in \vec{x}\,. \end{array}$$

This program transformation provides a simple implementation of the observational product of Def. **21**. Moreover, the logical abstract domains no longer need to perform purification.

Theorem 28. *A static analysis of the transformed program with a (reduced/iteratively reduced) product of logical abstract domains only involves purified formulæ hence can be performed componentwise (with reduction) without changing the observational semantics.*

Proof. The proof is by structural induction on programs. For the base cases, first consider the assignment $\mathtt{x} := \mathtt{e}$. For logical abstract domains which do not have $\mathtt{e}$ in their theory $\mathcal{T}_k$, the tranformer is $\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge x = \mathtt{e}[\mathtt{x} \leftarrow x']$, which is purified into $\exists y : \exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge x = y$, which is equivalent to $\exists x : \Psi$, so that we don't introduce any more variable to observe. For the other domains, $\mathtt{e}$ is pure so no purification is needed. The handling of backward assignement is similar and the test only contains boolean formulæ in a purified program so that no purification is needed there either. □

Purification can also be performed for non-disjoint theories, but this requires using as many variables as the number of theories that contain the expression $\mathtt{e}$ in their language, so that we can use existencials and remain precise by asserting the equality between thoses variables. The transformer will then be $\exists\, x'_i : P_i[\mathtt{x} \leftarrow x'_i] \wedge x''_i = \mathtt{e}[\mathtt{x} \leftarrow x'_i] \wedge \exists\, x'_j : P_j[\mathtt{x} \leftarrow x'_j] \wedge x''_j = \mathtt{e}[\mathtt{x} \leftarrow x'_j] \wedge \ldots \wedge \mathtt{x} = x''_i = x''_j \ldots$

7.4 Evolving reduced product

Despite constant progress in this area, SMT solvers seem to be too expensive for an extensive use on programs of realistic size. But on that aspect also, the reduced product approach can help: instead of a global refinement of a static analyzer, one can also consider local ones, e.g. when precision must be locally enhanced to prove the invariance of a user-provided assertion or a loop invariant. In that case the reduced product can evolve locally to include a new abstract domain when more precision is required and to exclude the new abstract domain when it is no longer required. In such an *evolving reduced product* for local refinement

- the upgrade with a new abstract domain adds a new component in the product initialized by top/$\mathtt{tt}$ which is then reduced pairwise with the other abstract domains (thus introducing the known information to the new component);
- the downgrade consists in a pairwise reduction of the component of the new abstract domain with the other components followed by the suppression of this component.

On non-critical parts, less precise but more efficient algebraic abstract domains are used to infer the necessary properties to use in the next critical part.

This is complementary to the use of variable packs in relational abstract domains [10], which can be seen as an evolution of one component of the product. In both cases, this evolution of the abstraction during the static analysis can either be decided before the analysis (e.g. based on the program syntax and the user-provided assertions to be proved) or during the iteration of the analysis itself (based on the observation of a lack of precision for an upgrade or the ineffectiveness of an abstraction for adowngrade).

7.5 On the design of static analyzers by iterated reduction between logical and algebraic domains with evolving refinement

The design we propose to combine algebraic and logical abstract domains is the following:

- purify the program (Sect. **7.3**) according to the theories of the logical domains (and even to operators which are poorly approximated by some algebraic abstract domains),
- use independent formulæ for each theory,
- reduce between each domains after individual computation steps, including checking for consistent interpretations (Sect. **7.2**),
- only introduce each domain wherever necessary (Sect. **7.4**),
- and finaly check if properties hold on each component after purification of the properties.

This design mechanism will give more precise results in cases where formulæ have to be approximated, and faster sound results. In addition, this design pattern should be used to put together independent works on so-far distinct research areas of static analysis by abstract interpretation and program proofs by theorem provers.

8 Related work

SMT solvers have been used in abstract interpretation, e.g. to implement specific logical abstract domains such as uninterpreted functions [18] or to automatically design transformers in presence of a best abstraction [25].

Contrary to the logical abstract interpretation framework developed by [19, 33, 17] we do not assume that the behavior of the program is described by formulæ in the same theory as the theory of the logical abstract domain, which offers no soundness guarantee, but instead we give the semantics of the logical abstract domains with respect to a set of possible semantics which includes the possibility of a sound combination of a mathematical semantics and a machine semantics, which is hard to achieve in SMT solvers without breaking down their performances (e.g. by encoding modular arithmetics in integer arithmetics or encoding floats either bitwise or with reals and roundings). So, our approach allows the description of the abstraction mechanism, comparisons of logical abstract domains, and to provide proofs of soundness on a formal basis.

Specific combinations of theories have been proposed for static analysis such as linear arithmetic and uninterpreted functions [19], universally quantified formulæ over theories such as linear arithmetic and uninterpreted functions [17] or the combination of a shape analysis with a numerical analysis [16][5]. The framework that we propose to combine algebraic and logical abstract domains can be used to design static analyzers incrementally, with minimal efforts to include new abstractions to improve precision either globally for the whole program analysis or locally, e.g. to prove loop invariants provided by the end user.

[5] These approaches can be formalized as observational reduced products.

9 Conclusion

We have proposed a new design method of static analyzers based on the reduced product or its approximation by the iterated reduction of the product to combine algebraic and logical abstract domains. This is for invariance inference but is also applicable to invariant verification. The key points were to consider an observational semantics with multiple interpretations and the understanding of the Nelson-Oppen theory combination procedure and its followers as an iterated reduction of the product of theories so that algebraic and logical abstract domains can be symmetrically combined in a product either reduced or with iterated reduction. The interest of the (reduced) product in logical abstract interpretation is that the analysis for each theory can be separated, even when they are not disjoint, thus allowing for an effective use of dedicated SMT solvers for each of the components.

Logical abstract domains may not be very efficient but can be used for rapid prototyping and then implemented in algebraic form with efficient algorithms. Despite their high cost, logical abstract domains can also be very expressive and could therefore be used, at least locally, to enhance the precision of algebraic abstractions through an evolving product with iterated reduction. Combined with algebraic abstractions they can sometimes be made sound for the machine semantics.

Finally, having shown the similarity and complementarity of analysis by abstract interpretation and program proofs by theorem provers and SMT solvers, we hope that our framework will allow reuse and cooperations between developments in both communities.

References

[1] 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 *Infotech@Aerospace 2010*, pages AIAA 2010–3385. AIAA, 2010.

[2] A.R. Bradley and Z. Manna. *The Calculus of Computation, Decision procedures with Applications to Verification*. Springer, 2007.

[3] A.R. Bradley, Z. Manna, and H.B. Sipma. What's decidable about arrays? In *VMCAI*, LNCS 3855, pages 427–442. Springer, 2006.

[4] 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.

[5] P. Cousot. The calculational design of a generic abstract interpreter. In *Calculational System Design*, volume 173, pages 421–505. IOS Press, 1999.

[6] 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. ACM Press, 1977.

[7] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282. ACM Press, 1979.

[8] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[9] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP*, LNCS 631, pages 269–295. Springer, 1992.

[10] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The Astrée analyser. In *ESOP*, LNCS 3444, pages 21–30. Springer, 2005.

[11] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Combination of abstractions in the Astrée static analyzer. In *ASIAN*, LNCS 4435, pages 272–300. Springer, 2008.

[12] R. Cytron, J. Ferrante, B.K. Rosen, M.N. Wegman, and F.K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *TOPLAS*, 13(4):451–490, 1991.

[13] A. Deutsch. On determining lifetime and aliasing of dynamically allocated data in higher-order functional specifications. In *17th POPL*, pages 157–168. ACM Press, 1990.

[14] P. Ferrara, F. Logozzo, and M. Fähndrich. Safer unsafe code in .NET. In *OOPSLA*, pages 329–346. ACM Press, 2008.

[15] P. Granger. Improving the results of static analyses of programs by local decreasing iterations. In *FST & TCS*, LNCS 652, pages 68–79. Springer, 1992.

[16] S. Gulwani, T. Lev-Ami, and M. Sagiv. A combination framework for tracking partition sizes. In *36th POPL*, pages 239–251. ACM Press, 2009.

[17] S. Gulwani, B. McCloskey, and A. Tiwari. Lifting abstract interpreters to quantified logical domains. In *35th POPL*, pages 235–246. ACM Press, 2008.

[18] S. Gulwani and G.C. Necula. Path-sensitive analysis for linear arithmetic and uninterpreted functions. In *SAS*, LNCS 3148, pages 328–343. Springer, 2007.

[19] S. Gulwani and A. Tiwari. Combining abstract interpreters. In *PLDI*, pages 376–386. ACM Press, 2006.

[20] C.A.R. Hoare. Monitors: an operating system structuring concept. *CACM*, 17(10):549–557, 1974.

[21] A. Miné. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *LCTES*, pages 54–63. ACM Press, 2006.

[22] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

[23] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *TOPLAS*, 1(2):245–257, 1979.

[24] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Inf.*, 6:319–340, 1976.

[25] T.W. Reps, S. Sagiv, and G. Yorsh. Symbolic implementation of the best transformer. In *VMCAI*, LNCS 2937, pages 252–266. Springer, 2004.

[26] N. Shankar and Harald Rueß. Combining Shostak theories. In *Rewriting Techniques and Applications*, LNCS 2378, pages 1–18. Springer, 2002.

[27] R.E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, 1984.

[28] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[29] C. Tinelli and M.T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In *Frontiers of Combining Systems*, pages 103–120. Kluwer Academic Publishers, 1996.

[30] C. Tinelli and C. Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theor. Comput. Sci.*, 290(1):291–353, 2003.

[31] C. Tinelli and C.G. Zarba. Combining decision procedures for sorted theories. In *JELIA*, LNCS 3229, pages 641–653. Springer, 2004.

[32] P. Tinelli and C.G. Zarba. Combining non-stably infinite theories. *Electr. Notes Theor. Comput. Sci.*, 86(1), 2003.

[33] A. Tiwari and S. Gulwani. Logical interpretation: Static program analysis using theorem proving. In *Automated Deduction – CADE-21*, LNCS 4603, pages 147–166. Springer, 2007.