Theories, Solvers and Static Analysis by Abstract Interpretation

Patrick Cousot, Courant Institute of Mathematical Sciences, New York University and École Normale Supérieure & Inria, Paris
Radhia Cousot, École Normale Supérieure & Inria, Paris and Centre National de la Recherche Scientifique, Paris
Laurent Mauborgne, Instituto Madrileño de Estudios Avanzados, Madrid

The algebraic/model theoretic design of static analyzers uses abstract domains based on representations of properties and pre-calculated property transformers. It is very efficient. The logical/proof theoretic approach uses SMT solvers/theorem provers and computation of property transformers on-the-fly. It is very expressive. We propose to unify both approaches, so that they can be combined to reach the sweet spot best adapted to a specific application domain in the precision/cost spectrum. We first give a new formalization of the proof theoretic approach in the abstract interpretation framework, introducing a semantics based on multiple interpretations to deal with the soundness of such approaches. Then we describe how to combine them with any other abstract interpretation-based analysis using an iterated reduction to combine abstractions. The key observation is that the Nelson-Oppen procedure, which decides satisfiability in a combination of logical theories by exchanging equalities and disequalities, computes a reduced product (after the state is enhanced with some new "observations" corresponding to alien terms). By abandoning restrictions ensuring completeness (such as disjointness, convexity, stably-infiniteness, or shininess, *etc*) we can even broaden the application scope of logical abstractions for static analysis (which is incomplete anyway).

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Program Verification—*Formal methods, validation, assertion checkers*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Mechanical verification, assertions, invariants*; F.3.2 [**Logics and Meaning of Programs**]: Semantics of Programming Languages—*Program analysis*

General Terms: Languages, Performance, Reliability, Theory, Verification

Additional Key Words and Phrases: Abstract Interpretation, Decision Procedures, Program Logics, Program Verification, SAT Modulo Theory, SMT solver, Semantics, Static Analysis, Theorem Proving

1. INTRODUCTION

Program verification, where the inductive argument necessary for the proof is either provided by the end-user or by refinement of the specification, typically use SMT solvers or theorem provers [Bradley and Manna 2007]. Recent progress in such techniques allowed their exploitation for static analysis by abstract interpretation [Cousot and Cousot 1977; Cousot and Cousot 1979c], (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 [Tiwari and Gulwani 2007; Gulwani et al. 2008]. But, because of efficiency restrictions of SMT solvers and theorem provers, these analyzers hardly scale up beyond small programs. Moreover, because of effectiveness restrictions of SMT solvers and theorem provers, their soundness proofs rest on a math-

– Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012, USA
– École normale supérieure, 45 rue d'Ulm, 75230 Paris cedex 05, Paris, France
– Fundación IMDEA Software, Facultad de Informática (UPM), Campus Montegancedo, 28660-Boadilla del Monte,, Madrid, Spain

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

ematical semantics that significantly differs from the implementation semantics of the programming language. For example, the theory of integer arithmetic is considered instead of modular arithmetic 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 arithmetic could be encoded with integer arithmetic and floats with rationals or bitwise, but then the performance of SMT solvers and theorem provers would be significatively degraded. On the other end, static analyzers such as Astrée [Bertrane et al. 2010; Cousot et al. 2005] that are based on algebraic abstractions of the machine semantics do not have such scalability, efficiency and soundness limitations. However, their expressiveness is limited by that of their abstract domains. It is therefore interesting not only to use SMT solvers and theorem provers in logical abstract domains but to combine algebraic and logical abstract interpretations to get the best of both worlds *i.e.* scalability, expressiveness, natural interface with the end-user using logical formulæ, and soundness with respect to the machine semantics. The proposed combination is based on the iterated reduced product [Cousot and Cousot 1979c], which is commonly used in algebraic abstract interpreters (e.g. in Astrée [Cousot et al. 2008]) while logical abstract interpreters use the Nelson-Oppen procedure [Nelson and Oppen 1979] to combine (disjoint, convex, stably-infinite) theories. The key new idea is to show that the 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 logical abstract domains, within algebraic abstract domains, and also between the logical and the algebraic abstract domains, including the case of abstractions evolving during the analysis.

We recall in section **2** the syntax, interpretation, satisfiability, validity, decidability, and comparison of first-order logical theories. In section **3**, we define the the mono-interpreted and *multi-interpreted* concrete semantics of programs [Cousot et al. 2010]. Section **4** introduces the basic notions of abstract interpretation. Section **5** suggests possible abstractions of the multi-interpreted semantics of programs. Section **6** defines the multi-interpreted semantics of first-order formulæ and the axiomatic semantics of programs modulo a multi-interpretation, a necessary concept to describe the soundness and relative precision of the *logical abstract domains* defined in section **7**. Next section **8** introduces *observational semantics*, which is a new construction generalizing static analysis practices and is necessary to describe the first step of the Nelson-Oppen procedure in the abstract interpretation framework. Section **9** recalls the notions of Cartesian and reduced product. Section **10** introduces iterated reduction, which can be used to implement the reduced product by reduction of the Cartesian product. In particular section **10.3** contains new incompleteness results on pairwise reductions and sufficient conditions for completeness. Then section **11** is focused on the Nelson-Oppen procedure and the links with the abstract interpretation. Finally, section **12** develops new methods to combine classical abstract interpretation and theorem provers. A comparison with related work is provided in section **13**, and the conclusion in section **14** proposes future work.

2. TERMINOLOGY FOR FIRST-ORDER LOGIC, THEORIES, INTERPRETATIONS AND MODELS

We use classical set-theoretical and predicate calculus notations [Monk 1969], see also [Chang and Keisler 1990; Mendelson 1997; Poizat 2000].

2.1. First-Order Logic

The set $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ of first-order formulæ on variables $\mathbb{x}$ and a signature $\langle \mathbb{f}, \mathbb{p} \rangle$ (where $\mathbb{f}$ are the function symbols, and $\mathbb{p}$ the predicate symbols such that $\mathbb{f} \cap \mathbb{p} = \emptyset$), is defined as:

$\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots \in \mathbb{x}$	variables
$\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots \in \mathbb{f}^0$	constants
$\mathtt{f}, \mathtt{g}, \mathtt{h}, \ldots \in \mathbb{f}^n, \quad \mathbb{f} \triangleq \bigcup_{n \geqslant 0} \mathbb{f}^n$	function symbols of arity $n \geqslant 1$

$$
\begin{array}{rcll}
t \in \mathbb{T}(\mathbb{x},\mathbb{f}) & & t ::= \mathtt{x} \mid \mathtt{c} \mid \mathtt{f}(t_1,\ldots,t_n) & \text{terms}\\
\mathtt{p},\mathtt{q},\mathtt{r},\ldots \in \mathbb{p}^n, & \mathbb{p}^0 \triangleq \{\mathtt{ff},\mathtt{tt}\}, & \mathbb{p} \triangleq \bigcup_{n\geqslant 0}\mathbb{p}^n & \text{predicate symbols of arity } n \geqslant 0,\\
a \in \mathbb{A}(\mathbb{x},\mathbb{f},\mathbb{p}) & & a ::= \mathtt{ff} \mid \mathtt{p}(t_1,\ldots,t_n) \mid \neg a & \text{atomic formulæ}\\
\varphi \in \mathbb{C}(\mathbb{x},\mathbb{f},\mathbb{p}) & & \varphi ::= a \mid \varphi \wedge \varphi & \text{clauses in simple conjunctive normal form}\\
e \in \mathbb{E}(\mathbb{x},\mathbb{f},\mathbb{p}) & \triangleq & \mathbb{T}(\mathbb{x},\mathbb{f}) \cup \mathbb{C}(\mathbb{x},\mathbb{f},\mathbb{p}) & \text{program expressions}\\
\Psi \in \mathbb{F}(\mathbb{x},\mathbb{f},\mathbb{p}) & & \Psi ::= a \mid \neg\Psi \mid \Psi \wedge \Psi \mid \exists \mathtt{x} : \Psi & \text{quantified first-order formulæ}
\end{array}
$$

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

2.2. Theories

The set $\vec{\mathtt{x}}_\Psi$ of *free variables* of a formula Ψ is defined inductively as the set of variables in the formula that are not in the scope of an existential quantifier. A *sentence* of $\mathbb{F}(\mathbb{x},\mathbb{f},\mathbb{p})$ is a formula with no free variable. A *theory* is a set of sentences [Chang and Keisler 1990] (called the *theorems* of the theory) and a signature, which must contain at least all the predicates and function symbols that appear in the theorems. The *language* of a theory is the set of quantified first-order formulæ that contain no predicate or function symbol outside of the signature of the theory.

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

2.3. Interpretations

This is better explained with the notion of interpretation of formulæ: An *interpretation* I for a signature $\langle \mathbb{f},\ \mathbb{p}\rangle$ is the pair $\langle I_{\mathcal{V}},\ I_\gamma\rangle$ such that

— $I_{\mathcal{V}}$ is a non-empty set of values,
— $\forall \mathtt{c} \in \mathbb{f}^0 : I_\gamma(\mathtt{c}) \in I_{\mathcal{V}}$ and $\forall n \geqslant 1 : \forall \mathtt{f} \in \mathbb{f}^n : I_\gamma(\mathtt{f}) \in I_{\mathcal{V}}^n \to I_{\mathcal{V}}$ interpret functions, and
— $\forall n \geqslant 0 : \forall \mathtt{p} \in \mathbb{p}^n : I_\gamma(\mathtt{p}) \in I_{\mathcal{V}}^n \to \mathcal{B}$ interprets predicates,

where $\mathcal{B} \triangleq \{\mathit{false},\mathit{true}\}$ are the Booleans. Let $\mathfrak{I}$ be the class of all such interpretations I. In a given interpretation $I \in \mathfrak{I}$, an *environment* [1] is a function from variables to values

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

We note $\eta[\mathtt{x} \leftarrow v]$ for the *assignment* of v to $\mathtt{x}$ in η such that $\eta[\mathtt{x} \leftarrow v](\mathtt{x}) \triangleq v$ and $\eta[\mathtt{x} \leftarrow v](\mathtt{y}) \triangleq \eta(\mathtt{y})$ when $\mathtt{x} \neq \mathtt{y}$.

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

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

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

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

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

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

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

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

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

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

2.4. Models

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

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

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

Quite often, the set of sentences of a theory is not defined extensively, but using a (generally finite) set of axioms that generates the set of theorems of the theory by implication. A theory is said to be *deductive* iff it is closed by deduction $\forall\Psi \in \mathcal{T} : \forall\Psi' \in \mathbb{F}(\mathbb{x},\mathbb{f},\mathbb{p}), \Psi \Rightarrow \Psi'$ implies $\Psi' \in \mathcal{T}$, that is all theorems that are true in all models of the theory are in the theory.

Let us recall that, by Gödel's compactness theorem, a first-order theory has a model if and only if every finite subset of it has a model and, by the Löwenheim-Skolem-Tarski theorem, no countable first-order theory with an infinite model can have exactly one model up to isomorphism [Poizat 2000].

The *theory* $\mathfrak{T}(I)$ *of an interpretation* I is the set $\mathfrak{T}(I) \triangleq \{\Psi \mid \exists\eta : I \models_\eta \Psi\}$ of sentences Ψ such that I is a model of Ψ. Such a theory is trivially deductive and satisfiable (*i.e.* has at least one model).

2.5. Satisfiability and Validity (Modulo Interpretations and Theory)

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

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

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

The four concepts can be extended to theories: a theory is satisfiable [2] (valid) if one (all) of the interpretations is a (are) model(s) of the theory *i.e.* $\mathfrak{M}(\mathcal{T}) \neq \emptyset$ (resp. $\mathfrak{M}(\mathcal{T}) = \mathfrak{I}$), and a theory is unsatisfiable (invalid) if all (one) of the interpretations make(s) one of the theorems of the theory false.

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

2.6. Decidable Theories

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

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

$$\mathsf{decide}_{\mathcal{T}}(\exists \vec{\mathbf{x}}_\Psi : \Psi) \;\Rightarrow\; \mathsf{satisfiable}_{\mathcal{T}}(\Psi) \qquad (\text{when } \mathcal{T} \text{ is satisfiable}) \tag{1}$$

Proof of (1).

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

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

The same result holds for validity:

$$\mathsf{decide}_{\mathcal{T}}(\forall \vec{\mathbf{x}}_\Psi : \Psi) \;\Rightarrow\; \mathsf{valid}_{\mathcal{T}}(\Psi) \tag{2}$$

Proof of (2).

$$\begin{array}{lll}
& \mathsf{decide}_{\mathcal{T}}(\forall \vec{\mathbf{x}}_\Psi : \Psi) & \\
\Leftrightarrow & (\forall \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T} & \text{⟅def. decision procedure⟆} \\
\Rightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi & \text{⟅def. } \mathfrak{M}(\mathcal{T}) \triangleq \{I \in \mathfrak{I} \mid \forall \Psi' \in \mathcal{T} : \exists \eta' : I \models_{\eta'} \Psi'\}\text{⟆} \\
\Leftrightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi & \\
& & \text{⟅since } \forall \vec{\mathbf{x}}_\Psi : \Psi \text{ has no free variable, } I \models_\eta \forall \vec{\mathbf{x}}_\Psi : \Psi \text{ does not depend on } \eta\text{⟆} \\
\Leftrightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \Psi & \text{⟅def. } I \models_\eta \forall \mathbf{x} : \Psi \text{ in section } \mathbf{2.3}\text{⟆} \\
\Leftrightarrow & \mathsf{valid}_{\mathcal{T}}(\Psi) & \text{⟅}\mathsf{valid}_{\mathcal{T}}(\Psi) \triangleq \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \Psi\text{⟆} \quad \square
\end{array}$$

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

$$\mathsf{valid}_{\mathcal{T}}(\Psi) \;\Leftrightarrow\; \mathsf{decide}_{\mathcal{T}}(\forall \vec{\mathbf{x}}_\Psi : \Psi) \qquad (\text{when } \mathcal{T} \text{ is decidable and deductive}) \tag{3}$$

Proof of (3). $\mathcal{T}$ is deductive, hence all valid sentences are theorems of the theory, so if $\mathsf{valid}_{\mathcal{T}}(\Psi)$ then $\forall \vec{\mathbf{x}}_\Psi : \Psi$ is a valid sentence of $\mathcal{T}$ and so it is in $\mathcal{T}$. $\square$

From that, we can obtain satisfiability of any formula:

$$\mathsf{satisfiable}_{\mathcal{T}}(\Psi) \;\Leftrightarrow\; \neg\,(\mathsf{decide}_{\mathcal{T}}(\forall \vec{\mathbf{x}}_\Psi : \neg\Psi)) \qquad (\text{when } \mathcal{T} \text{ is decidable and deductive}) \tag{4}$$

Proof of (4).

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

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

$$\mathsf{satisfiable}_{\mathcal{T}}(\Psi) \;\Leftrightarrow\; (\mathsf{decide}_{\mathcal{T}}(\exists \vec{\mathbf{x}}_\Psi : \Psi)) \qquad \text{(when } \mathcal{T} \text{ is decidable and complete)} \qquad (5)$$

Proof of (5). Assume $\mathcal{T}$ is complete. Then, either $\exists \vec{\mathbf{x}}_\Psi : \Psi \in \mathcal{T}$, in which case $\mathsf{decide}_{\mathcal{T}}(\exists \vec{\mathbf{x}}_\Psi : \Psi)$ returns *true* and we conclude $\mathsf{satisfiable}_{\mathcal{T}}(\Psi)$. Or $\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T}$ so $\mathsf{decide}_{\mathcal{T}}(\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi))$ returns *true*. But if a sentence is in the theory, that means that for all models of that theory, the sentence is true, so:

$$\begin{array}{lll}
& \neg\mathsf{decide}_{\mathcal{T}}(\exists \vec{\mathbf{x}}_\Psi : \Psi) & \\
\Leftrightarrow & \mathsf{decide}_{\mathcal{T}}(\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi)) & \text{⟅}\mathcal{T}\text{ complete⟆} \\
\Leftrightarrow & \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) \in \mathcal{T} & \text{⟅def. decision procedure⟆} \\
\Rightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) & \text{⟅def. } \mathfrak{M}(\mathcal{T}') \triangleq \{I \in \mathfrak{I} \mid \forall \Psi' \in \mathcal{T}' : \exists \eta' : I \models_{\eta'} \Psi'\}\text{⟆} \\
\Rightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \forall \eta : I \models_\eta \neg(\exists \vec{\mathbf{x}}_\Psi : \Psi) & \text{⟅}\neg(\exists \vec{\mathbf{x}}_\Psi : \Psi)\text{ has no free variable⟆} \\
\Leftrightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \neg(\exists \eta : I \models_\eta \exists \vec{\mathbf{x}}_\Psi : \Psi) & \text{⟅def. }\neg\text{⟆} \\
\Leftrightarrow & \forall I \in \mathfrak{M}(\mathcal{T}) : \neg(\exists \eta : I \models_\eta \Psi) & \text{⟅def. } I \models_\eta \exists \mathbf{x} : \Psi \text{ in section } \mathbf{2.3}\text{⟆} \\
\Leftrightarrow & \neg(\exists I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi) & \text{⟅def. }\neg\text{⟆} \\
\Leftrightarrow & \neg\mathsf{satisfiable}_{\mathcal{T}}(\Psi) & \text{⟅def. } \mathsf{satisfiable}_{\mathcal{T}}(\Psi) \triangleq \exists I \in \mathfrak{M}(\mathcal{T}) : \exists \eta : I \models_\eta \Psi\text{⟆} \quad \square
\end{array}$$

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

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

$$\begin{array}{l}
\mathsf{decide}_{\mathcal{T}}(\exists \vec{\mathbf{x}}_{\Psi\wedge\neg\Psi'} : \Psi \wedge \neg\Psi') \;\Rightarrow\; \neg\mathsf{valid}_{\mathcal{T}}(\forall \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \Rightarrow \Psi') \\
\qquad\qquad (\Leftrightarrow \text{ when } \mathcal{T} \text{ is complete for } \exists \vec{\mathbf{x}}_{\Psi\wedge\neg\Psi'} : \Psi \wedge \neg\Psi') \qquad (6)
\end{array}$$

Proof of (6).

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

2.7. Comparison of Theories

Except for decision procedures, theories are equivalent when they have the same models. A theory $\mathcal{T}_1$ is *more general* than a theory $\mathcal{T}_2$ when all models of $\mathcal{T}_2$ are models of $\mathcal{T}_1$ *i.e.* $\mathfrak{M}(\mathcal{T}_2) \subseteq \mathfrak{M}(\mathcal{T}_1)$. A sufficient condition for $\mathcal{T}_1$ to be more general than $\mathcal{T}_2$ is $\mathcal{T}_1 \subseteq \mathcal{T}_2$ (since $\mathcal{T}_1 \subseteq \mathcal{T}_2$ implies $\mathfrak{M}(\mathcal{T}_2) \subseteq \mathfrak{M}(\mathcal{T}_1)$). The converse holds for deductive theories. The most general theory for a given

signature is the theory $\{\mathtt{tt}\}$ (or equivalently its deductive closure), also called the *theory of logical validities*. If a theory $\mathcal{T}_1$ is more general than $\mathcal{T}_2$, then we have, for all formulæ Ψ:

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

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

The *combination* of two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ is $\mathcal{T}_1 \cup \mathcal{T}_2$. In other words, the combination of two theories is the theory that satisfy the theorems of both theories. Thus, the models of the combination of two theories are models for each of them. The combination of two satisfiable theories is not always satisfiable: it is possible that no model of the first theory satisfies all theorems of the second one. In [Tinelli and Harandi 1996, Cor. 3.3], a sufficient condition for the combination of two satisfiable theories to be satisfiable is described: if the two theories have disjoint signatures [3] and they both have an infinite model, then their combination is satisfiable.

3. CONCRETE SEMANTICS

Abstractions in abstract interpretation [Cousot and Cousot 1977; Cousot and Cousot 1979c], are relative to a concrete semantics specifying the possible executions of programs of a programming language. The concrete semantics of a language can be defined with respect to a single interpretation of symbols (e.g. the mathematical semantics of a specification language with integers and reals) or with respect to several possible interpretations of symbols (e.g. the float semantics of a programming language with different memory word sizes and rounding modes). Although the classical mono-interpreted semantics is a particular case of multi-interpreted semantics, both cases are considered in sections **3.2** and **3.3**.

3.1. Programs

We let $\mathbb{P}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ be the set of programs $\mathtt{P}$ over a signature $\langle \mathbb{x},\ \mathbb{f},\ \mathbb{p} \rangle$.

Example 3.1 (*Imperative programs*). In our examples, we consider an imperative programming language on a given signature $\langle \mathbb{x},\ \mathbb{f},\ \mathbb{p} \rangle$. Programs are built out of basic expressions $e \in \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ and imperative commands including assignments $\mathtt{x} := e$ and tests/guards φ appear in conditionals and loops whose syntax, as well as that of programs, is irrelevant.

$$\mathtt{P}, \ldots \in \mathbb{P}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \qquad \mathtt{P} ::= \mathtt{x} := e \mid \varphi \mid \ldots \qquad \text{programs} \qquad \blacksquare$$

Programs are usually intended to be mono-interpreted (section **3.2**), that is to have a unique interpretation (defining e.g. their possible executions on an abstract machine). However, programs are often multi-interpreted (section **3.3**), since their executions may vary on different machines or may differ from their mathematical semantics.

3.2. Mono-Interpreted Concrete Semantics

A *mono-interpreted concrete semantics* $C_{\mathfrak{I}}[\![\mathtt{P}]\!]$ of programs $\mathtt{P}$ as defined by a program interpretation $\mathfrak{I} \in \mathfrak{J}$. Most often (e.g. when looking for safety properties), we can define that concrete semantics as a set of post-fixpoints in a complete lattice/cpo of concrete properties $\langle \mathcal{P}_{\mathfrak{I}},\ \subseteq \rangle$ and a concrete transformer $F_{\mathfrak{I}}[\![\mathtt{P}]\!]$. We define $\mathbf{postfp}^{\leq} f \triangleq \left\{ x \;\middle|\; f(x) \leq x \right\}$.

$$\begin{array}{rcll} & & \mathcal{R}_{\mathfrak{I}} & \text{concrete observables}[4] \\ & \mathcal{P}_{\mathfrak{I}} & \triangleq \wp(\mathcal{R}_{\mathfrak{I}}) & \text{concrete properties } [5] \\ & F_{\mathfrak{I}}[\![\mathtt{P}]\!] & \in \mathcal{P}_{\mathfrak{I}} \to \mathcal{P}_{\mathfrak{I}} & \text{concrete transformer of program } \mathtt{P} \\ C_{\mathfrak{I}}[\![\mathtt{P}]\!] & \triangleq \mathbf{postfp}^{\subseteq} F_{\mathfrak{I}}[\![\mathtt{P}]\!] & \in \wp(\mathcal{P}_{\mathfrak{I}}) & \text{concrete semantics of program } \mathtt{P}. \end{array}$$

[3] $\mathbb{F}(\mathbb{x}, \mathbb{f}_1, \mathbb{p}_1)$ and $\mathbb{F}(\mathbb{x}, \mathbb{f}_2, \mathbb{p}_2)$ such that $(\mathbb{f}_1 \cup \mathbb{p}_1) \cap (\mathbb{f}_2 \cup \mathbb{p}_2) = \{=\}$ and all equalities in common have the same interpretation.

Example 3.2. In the context of [Floyd 1967]'s method to prove invariance properties for imperative languages with program interpretation $\mathfrak{I} \in \mathfrak{Z}$, we can take a concrete state to be a function from variables [6] to elements in the set $\mathfrak{I}_\mathcal{V}$, so that properties, that is global invariants, are sets of such functions.

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

The concrete transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$ of program P defines the (set-theoretic version of the) verification condition $F_\mathfrak{I}[\![\mathtt{P}]\!](S) \subseteq S$ for $S \in \mathcal{R}_\mathfrak{I}$ to be a program inductive invariant (assigning possible values to program variables at each program point). This concrete transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$ is defined by structural induction on the program P in terms of the complete lattice operations $\langle \wp(\mathcal{R}_\mathfrak{I}), \subseteq, \emptyset, \mathcal{R}_\mathfrak{I}, \cup, \cap \rangle$ and the following local invariance transformers

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

■

Note that if the concrete transformer admits a least fixpoint, then it is enough to consider only that least fixpoint and we do not need to compute the entire set of post-fixpoints (see also section **4.3**).

Example 3.3 (*Least fixpoint concrete semantics*). $\langle \mathcal{P}_\mathfrak{I}, \subseteq, \emptyset, \mathcal{R}_\mathfrak{I}, \cup, \cap \rangle$ is a complete lattice so if the transformer $F_\mathfrak{I}[\![\mathtt{P}]\!]$ is increasing then, by [Tarski 1955], we have $\mathbf{lfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!] = \bigcap \mathbf{postfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!] \in \mathbf{postfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!]$ which is the strongest post-fixpoint. Notice that all $\{P \in \mathcal{P}_\mathfrak{I} \mid \mathbf{lfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!] \subseteq P\}$ are all valid program properties but only the elements of $\mathbf{postfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!]$ are inductive properties as needed for proofs. This is the case in example *3.2*, where $\mathbf{lfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!]$ defines the strongest invariant for the Floyd program proof method [Floyd 1967]. ■

Example 3.4. The program $\mathtt{P} \triangleq \mathtt{x=1;\ while\ true\ \{x=incr(x)\}}$ with the arithmetic interpretation $\mathfrak{I}$ on integers $\mathfrak{I}_\mathcal{V} = \mathbb{Z}$ has loop invariant $\mathbf{lfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!]$ where $F_\mathfrak{I}[\![\mathtt{P}]\!](X) \triangleq \{\eta \in \mathcal{R}_\mathfrak{I} \mid \eta(\mathtt{x}) = 1\} \cup \{\eta[\mathtt{x} \leftarrow \eta(\mathtt{x}) + 1] \mid \eta \in X\}$. The increasing chain of iterates $F_\mathfrak{I}[\![\mathtt{P}]\!]^n = \{\eta \in \mathcal{R}_\mathfrak{I} \mid 0 < \eta(\mathtt{x}) < n\}$ has the limit $\mathbf{lfp}^\subseteq F_\mathfrak{I}[\![\mathtt{P}]\!] = \bigcup_{n \geqslant 0} F_\mathfrak{I}[\![\mathtt{P}]\!]^n = \{\eta \in \mathcal{R}_\mathfrak{I} \mid 0 < \eta(\mathtt{x})\}$. ■

If the concrete transformer of a program has no least fixpoint, the entire set of post-fixpoints is defined by the concrete semantics although only one of them is needed for a given proof (because the concrete semantics defines all the possible ways to make proofs).

3.3. Multi-Interpreted Concrete Semantics

A *multi-interpreted concrete semantics*, provides a semantics for programs P in the context of a set of interpretations $\mathcal{I} \in \wp(\mathfrak{Z})$. Then a program property in $\mathcal{P}_\mathcal{I}$ provides, for each interpretation in $\mathcal{I}$, a set of program observables satisfying that property in that interpretation.

$$\begin{array}{lll} \mathcal{R}_I & & \text{program observables under the interpretation } I \in \mathcal{I} \\ \mathcal{P}_\mathcal{I} & \triangleq & I \in \mathcal{I} \mathrel{\not\rightarrow} \wp(\mathcal{R}_I) \quad \text{interpreted properties under the set of interpretations } \mathcal{I} \\ & \simeq & \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\})\ [7] \end{array}$$

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

[4] Examples of observables are set of states, set of partial or complete execution traces, infinite/transfinite execution trees, *etc*.
[5] A property is understood as the set of elements satisfying this property.
[6] maybe including the program counter *etc*.
[7] A partial function $f \in A \twoheadrightarrow B$ with domain $\mathsf{dom}(f) \in \wp(A)$ is understood as the relation $\{\langle x, f(x)\rangle \in A \times B \mid x \in \mathsf{dom}(f)\}$ and maps $x \in A$ to $f(x) \in B$, written $x \in A \mathrel{\not\rightarrow} f(x) \in B$ or $x \in A \mathrel{\not\rightarrow} B_x$ when $\forall x \in A : f(x) \in B_s \subseteq B$.

$$\begin{array}{rcll} F_{\mathcal{I}}[\![\mathtt{P}]\!] & \in & \mathcal{P}_{\mathcal{I}} \to \mathcal{P}_{\mathcal{I}} & \text{multi-interpreted concrete transformer of program P} \\ & \triangleq & \lambda\, P \in \mathcal{P}_{\mathcal{I}} \bullet \lambda\, I \in \mathcal{I} \bullet F_I[\![\mathtt{P}]\!](P(I)) & \\ C_{\mathcal{I}}[\![\mathtt{P}]\!] & \in & \wp(\mathcal{P}_{\mathcal{I}}) & \text{multi-interpreted concrete semantics} \\ & \triangleq & \mathbf{postfp}^{\dot{\subseteq}}\, F_{\mathcal{I}}[\![\mathtt{P}]\!] & \end{array}$$

where $\dot{\subseteq}$ is the pointwise subset ordering.

Example 3.5. In the context of invariance properties for imperative languages with multiple program interpretations $\mathcal{I} \in \wp(\mathfrak{I})$, example *3.2* can be generalized by taking

$$\mathcal{R}_I \triangleq \mathbb{x} \to I_{\mathcal{V}} \quad \text{concrete interpreted environments for interpretation } I \in \mathcal{I}.$$

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

$$\begin{array}{rcll} \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P & \triangleq & \{\langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \mid I \in \mathcal{I} \wedge \langle I,\, \eta\rangle \in P)\} & \text{assignment post-condition} \\ \mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]P & \triangleq & \{\langle I,\, \eta\rangle \mid I \in \mathcal{I} \wedge \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \in P\} & \text{assignment pre-condition} \\ \mathsf{p}_{\mathcal{I}}[\![\varphi]\!]P & \triangleq & \{\langle I,\, \eta\rangle \in P \mid I \in \mathcal{I} \wedge [\![\varphi]\!]_I\eta = \mathit{true}\} & \text{test.} \end{array} \tag{7}$$

In particular for $\mathcal{I} = \{\mathfrak{I}\}$, we get the transformers of example *3.2*, up to the isomorphism $\iota_{\mathfrak{I}}(P) \triangleq \{\langle \mathfrak{I},\, \eta\rangle \mid \eta \in P\}$ with inverse $\iota_{\mathfrak{I}}^{-1}(Q) \triangleq \{\eta \mid \langle \mathfrak{I},\, \eta\rangle \in Q\}$. Observe that the transformers are complete morphisms for union and intersection and so are increasing for the subset ordering. In general, it follows that the transformer $F_{\mathcal{I}}[\![\mathtt{P}]\!]$ for the invariance semantics has the same properties. ∎

The natural ordering to express abstraction (or precision) on multi-interpreted semantics is the subset ordering, which gives a lattice structure to the set of multi-interpreted properties: a property P_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, we can express systematic abstractions of the multi-interpreted semantics in section **5**. But first we will recall the foundations of static analysis of program properties by abstract interpretation in section **4**.

4. BACKGROUND ON ABSTRACT INTERPRETATION

4.1. Abstract Domains

In static analysis by abstract interpretation [Cousot and Cousot 1977; Cousot and Cousot 1979c], abstract domains are used to encapsulate abstract program properties and abstract operations (including the logical lattice structure, elementary transformers, convergence acceleration operators, *etc*.).

Example 4.1. Typically, an abstract domain for an imperative language would be a tuple

$$\langle A, \sqsubseteq, \bot, \top, \sqcup, \sqcap, \nabla, \Delta, \bar{\mathsf{f}}, \bar{\mathsf{b}}, \bar{\mathsf{p}}, \ldots \rangle$$

where

$$\begin{array}{rcll} \overline{P}, \overline{Q}, \ldots & \in & A & \text{abstract properties} \\ \sqsubseteq & \in & A \times A \to \mathcal{B} & \text{abstract partial order}\,[8] \\ \bot, \top & \in & A & \text{infimum, supremum } (\forall \overline{P} \in A : \bot \sqsubseteq \overline{P} \sqsubseteq \top) \\ \sqcup, \sqcap, \nabla, \Delta & \in & A \times A \to A & \text{abstract join, meet, widening, narrowing} \\ & \ldots & & \\ \bar{\mathsf{f}} & \in & (\mathbb{x} \times \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \to A \to A & \text{abstract forward assignment transformer} \\ \bar{\mathsf{b}} & \in & (\mathbb{x} \times \mathbb{E}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \to A \to A & \text{abstract backward assignment transformer} \\ \bar{\mathsf{p}} & \in & \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to A \to A & \text{abstract condition transformer.} \end{array}$$

A procedural language would include projection (to handle procedure calls) and the analysis of a higher-order functional language would require the domain to provide an operation to abstract partial application of a function to a subset of its arguments. ∎

4.2. Abstract Semantics

The abstract semantics $\overline{C}[\![\mathtt{P}]\!] \in \wp(A)$ of a program P is assumed to be given as a set of post-fixpoints $\overline{C}[\![\mathtt{P}]\!] \triangleq \{\overline{P} \mid \overline{F}[\![\mathtt{P}]\!](\overline{P}) \sqsubseteq \overline{P}\}$ or in least fixpoint form $\overline{C}[\![\mathtt{P}]\!] \triangleq \{\mathbf{lfp}^{\sqsubseteq}\, \overline{F}[\![\mathtt{P}]\!]\}$ (or, by the singleton isomorphism, the more frequent $\mathbf{lfp}^{\sqsubseteq}\, \overline{F}[\![\mathtt{P}]\!]$) when such a least fixpoint does exist (e.g. [Tarski 1955]) where $\overline{F}[\![\mathtt{P}]\!] \in A \to A$ is the abstract transformer of program P built out of the primitives $\bot, \top, \sqcup, \sqcap, \nabla, \Delta, \bar{\mathsf{f}}, \bar{\mathsf{b}}, \bar{\mathsf{p}}, \ldots$ [9]. As was the case for the concrete semantics, we preferably use least fixpoints where possible.

4.3. Soundness of Abstract Domains

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

$$\gamma \in A \xrightarrow{\nearrow} \mathcal{P}_{\mathfrak{I}} \quad \text{concretization}\ [10]$$

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

$$\begin{array}{llll} (\overline{P} \sqsubseteq \overline{Q}) \Rightarrow (\gamma(\overline{P}) \subseteq \gamma(\overline{Q})) & \text{order} & \gamma(\bot) = \emptyset & \text{infimum} \\ \gamma(\overline{P} \sqcup \overline{Q}) \supseteq (\gamma(\overline{P}) \cup \gamma(\overline{Q})) & \text{join} & \gamma(\top) = \top_{\mathfrak{I}} & \text{supremum}\ [11] \\ \ldots & & & \end{array}$$

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

4.4. Soundness of Abstract Semantics

Definition 4.2 (Soundness and completeness of abstract semantics). *The abstract semantics $\overline{C}[\![P]\!] \in \wp(A)$ for an abstract domain $\langle A, \sqsubseteq\rangle$ of a program P is* sound *with respect to a concrete semantics $C[\![P]\!] \in \wp(C)$ for a concrete domain $\langle C, \leqslant\rangle$ and an increasing concretization $\gamma \in A \mapsto C$ whenever*

$$\forall \overline{P} \in A : (\exists \overline{C} \in \overline{C}[\![P]\!] : \ \overline{C} \sqsubseteq \overline{P}) \Rightarrow (\exists C \in C[\![P]\!] : \ C \leqslant \gamma(\overline{P})) \tag{8}$$

(so that any proof in the abstract can be done in the concrete). It is complete *whenever*

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

(so that any proof in the concrete of an abstract property can also be done directly in the abstract). ∎

Theorem 4.3 (Compositionality of abstractions). *The composition of sound (resp. complete) abstractions is sound (resp. complete).* □

[8] If $\sqsubseteq$ is a pre-order then A is assumed to be quotiented by the equivalence relation $\equiv \triangleq \sqsubseteq \cap \sqsubseteq^{-1}$.
[9] In general, this is more complex, with formulæ involving many fixpoints, but this simple setting already exhibits all difficulties.
[10] Given posets $\langle L, \sqsubseteq\rangle$ and $\langle P, \leqslant\rangle$, we let $L \xrightarrow{\nearrow} P$ to be the set of *increasing* (*isotone*, *monotone*, ...) maps f of L into P *i.e.* $\forall x, y \in L : x \sqsubseteq y$ implies $f(x) \leqslant f(y)$.
[11] For example, $\top_{\mathfrak{I}} \triangleq \mathcal{R}_{\mathfrak{I}}$ in the context of invariance properties for imperative languages in example *3.2*.

PROOF. Assume (8) respectively for $A_1, \sqsubseteq_1, C_1[\![\mathrm{P}]\!]$ and $A_2, \sqsubseteq_2, C_2$ with $\gamma_{21} \in A_2 \xrightarrow{\nearrow} A_1$ and $C_2[\![\mathrm{P}]\!]$ and $A_3, \sqsubseteq_3, C_3[\![\mathrm{P}]\!]$ with $\gamma_{32} \in A_3 \xrightarrow{\nearrow} A_2$. For all $\overline{P} \in A_3$, we have

$$\begin{array}{lll} & (\exists \overline{C}_3 \in C_3[\![\mathrm{P}]\!] : \ \overline{C}_3 \sqsubseteq_3 \overline{P}) & \\ \Rightarrow & (\exists \overline{C}_2 \in C_2[\![\mathrm{P}]\!] : \ \overline{C}_2 \sqsubseteq_2 \gamma_{32}(\overline{P})) & \wr \text{by (8) for } A_2, \sqsubseteq_2, C_2[\![\mathrm{P}]\!] \text{ and } A_3, \sqsubseteq_3, C_3[\![\mathrm{P}]\!] \text{ with } \gamma_{32} \wr \\ \Rightarrow & (\exists \overline{C}_1 \in C_1[\![\mathrm{P}]\!] : \ \overline{C}_1 \sqsubseteq_1 \gamma_{21} \circ \gamma_{32}(\overline{P})) & \wr \text{by (8) for } A_1, \sqsubseteq_1, C_1[\![\mathrm{P}]\!] \text{ and } A_2, \sqsubseteq_2, C_2[\![\mathrm{P}]\!] \text{ with } \gamma_{21} \wr \end{array}$$

proving (8) for $A_1, \sqsubseteq_1, C_1[\![\mathrm{P}]\!]$ and $A_3, \sqsubseteq_3, C_3[\![\mathrm{P}]\!]$ with $\gamma_{21} \circ \gamma_{32}$. The proof for completeness is similar. □

It follows from theorem 4.3 that the soundness (resp. completeness) of an abstract semantics with respect to the concrete semantics of section **3** can be proved directly or using the composition of intermediate abstractions.

When the concrete and abstract semantics are defined in post-fixpoint form, the soundness of the abstract semantics follows from the soundness of the abstraction in section **4.3** and the soundness of the abstract transformer [Cousot and Cousot 1977; Cousot and Cousot 1979c]

$$\forall \overline{P} \in A : F[\![\mathrm{P}]\!] \circ \gamma(\overline{P}) \leqslant \gamma \circ \overline{F}[\![\mathrm{P}]\!](\overline{P})^{12} \tag{9}$$

THEOREM 4.4 (SOUNDNESS OF AN ABSTRACT POST-FIXPOINT SEMANTICS). *If* $C[\![P]\!] \triangleq \boldsymbol{postfp}^{\leqslant} F[\![P]\!]$, $\overline{C}[\![P]\!] \triangleq \boldsymbol{postfp}^{\sqsubseteq} \overline{F}[\![P]\!]$ *and* $\gamma : A \rightarrow C$ *is increasing, then (9) implies (8).* □

PROOF. For all $\overline{P} \in A$, we have

$$\begin{array}{lll} & \exists \overline{C} \in \overline{C}[\![\mathrm{P}]\!] : \ \overline{C} \sqsubseteq \overline{P} & \\ \Rightarrow & \exists \overline{C} : \overline{F}[\![\mathrm{P}]\!](\overline{C}) \sqsubseteq \overline{C} \wedge \overline{C} \sqsubseteq \overline{P} & \wr \text{def. } \overline{C}[\![\mathrm{P}]\!] \triangleq \mathbf{postfp}^{\sqsubseteq} \overline{F}[\![\mathrm{P}]\!] = \{\overline{P} \mid \overline{F}[\![\mathrm{P}]\!](\overline{P}) \sqsubseteq \overline{P}\} \wr \\ \Rightarrow & \exists \overline{C} : \gamma(\overline{F}[\![\mathrm{P}]\!](\overline{C})) \leqslant \gamma(\overline{C}) \wedge \gamma(\overline{C}) \leqslant \gamma(\overline{P}) & \wr \gamma \text{ increasing} \wr \\ \Rightarrow & \exists \overline{C} : F[\![\mathrm{P}]\!](\gamma(\overline{C})) \leqslant \gamma(\overline{C}) \wedge \gamma(\overline{C}) \leqslant \gamma(\overline{P}) & \wr \text{by hypothesis (9), def. } \circ \text{, and transitivity} \wr \\ \Rightarrow & \exists C : F[\![\mathrm{P}]\!](C) \leqslant C \wedge C \leqslant \gamma(\overline{P}) & \wr \text{choosing } C = \gamma(\overline{C}) \wr \\ \Rightarrow & \exists C \in C[\![\mathrm{P}]\!] : \ C \leqslant \gamma(\overline{P}) & \wr \text{def. } C[\![\mathrm{P}]\!] \triangleq \mathbf{postfp}^{\leqslant} F[\![\mathrm{P}]\!] \triangleq \{C \mid F[\![\mathrm{P}]\!](C) \leqslant C\} \wr \end{array}$$ □

Example 4.5. Continuing example *3.2* in the context of invariance properties for imperative languages, the soundness of the abstract transformer generally follows from the following *local soundness conditions* on abstract transformers, for all $\overline{P} \in A$,

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

Observe that soundness is preserved by composition of increasing concretizations.

4.5. Iterates with Widening

When the abstract domain does not satisfy the ascending chain condition, a widening is needed both to cope with the absence of infinite disjunctions and to enforce the convergence of iterations to a post-fixpoint. Let us recall the following definitions and results [Cousot and Cousot 1976; Cousot and Cousot 1977; Cousot 1978].

DEFINITION 4.6 (WIDENING). *Let* $\langle A, \sqsubseteq \rangle$ *be a poset. Then an* over-approximating widening $\nabla \in A \times A \mapsto A$ *is such that*

[12] The composition of functions is defined such that $f \circ g(x) = f(g(x))$ where $x \in \mathsf{dom}(g)$ and $g(x) \in \mathsf{dom}(f)$.

(a) $\forall x, y \in A : x \sqsubseteq x \mathbin{\triangledown} y \wedge y \leqslant x \mathbin{\triangledown} y$[13].

A terminating widening $\triangledown \in A \times A \mapsto A$ *is such that*

(b) Given any sequence $\langle x^n,\ n \geqslant 0\rangle$*, the sequence* $y^0 = x^0, \ldots, y^{n+1} = y^n \mathbin{\triangledown} x^n, \ldots$ *converges, i.e.* $\exists \ell \in \mathbb{N} : \forall n \geqslant \ell : y^n = y^\ell$ *in which case* y^ℓ *is called the* limit *of the widened sequence* $\langle y^n,\ n \geqslant 0\rangle$.

Traditionally a widening *is considered to be both over-approximating and terminating.* ∎

Definition 4.7 (Iterates with widening). *The iterates of a transformer* $\overline{F}[\![P]\!] \in A \mapsto A$ *from the infimum* $\bot \in A$ *with widening* $\triangledown \in A \times A \mapsto A$ *in the poset* $\langle A, \sqsubseteq\rangle$ *are defined by recurrence as* $\overline{F}^0 = \bot$, $\overline{F}^{n+1} = \overline{F}^n$ *when* $\overline{F}[\![P]\!](\overline{F}^n) \sqsubseteq \overline{F}^n$ *and* $\overline{F}^{n+1} = \overline{F}^n \mathbin{\triangledown} \overline{F}[\![P]\!](\overline{F}^n)$ *otherwise.* ∎

Theorem 4.8 (Limit of the iterates with widening). *The iterates in a poset* $\langle A, \sqsubseteq, \bot\rangle$ *of a transformer* $\overline{F}[\![P]\!]$ *from the infimum* $\bot$ *with widening* $\triangledown$ *converge and their limit is a post-fixpoint of the transformer.* □

Proof. The assumption that the iterates diverge (that is $\forall n \in \mathbb{N} : \overline{F}^{n+1} \neq \overline{F}^n$) contradicts condition (b) of definition 4.6. By reductio ad absurdum, the limit $\overline{F}^\ell$ does exist. By definition 4.7, either $\overline{F}[\![\mathtt{P}]\!](\overline{F}^\ell) \sqsubseteq \overline{F}^\ell$ or else $\overline{F}^\ell = \overline{F}^{\ell+1} = \overline{F}^\ell \mathbin{\triangledown} \overline{F}[\![\mathtt{P}]\!](\overline{F}^\ell) \sqsupseteq \overline{F}[\![\mathtt{P}]\!](\overline{F}^\ell)$, by condition (a) of definition 4.6. In both cases, $\overline{F}^\ell \in \mathbf{postfp}^\sqsubseteq \overline{F}[\![\mathtt{P}]\!]$. □

4.6. Best Abstraction

Let us recall from [Cousot and Cousot 1979c] that if any concrete property $P \in \mathcal{P}_\mathfrak{I}$ has a best abstraction in the abstract domain $\langle A, \sqsubseteq\rangle$, we have a Galois connection $\langle \mathcal{P}_\mathfrak{I}, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$ such that, by definition, $\forall P \in \mathcal{P}_\mathfrak{I} : \forall \overline{P} \in A : \alpha(P) \sqsubseteq \overline{P} \Leftrightarrow P \subseteq \gamma(\overline{P})$. This implies that $\alpha(P)$ is a sound abstraction of P since $P \subseteq \gamma(\alpha(P))$. Moreover $\alpha(P)$ is the best sound abstraction of P since if $\overline{P}$ is another sound abstraction of P then $P \subseteq \gamma(\overline{P})$ which implies $\alpha(P) \sqsubseteq \overline{P}$ and so $\alpha(P)$ is more precise than $\overline{P}$ in the abstract (and so also in the concrete since γ is increasing). Moreover the abstraction α preserves existing least upper bounds and so is increasing, *i.e.* preserves the concrete implication $\subseteq$ and, by duality, γ preserves existing greatest lower bounds and so is increasing, *i.e.* preserves the abstract implication $\sqsubseteq$. We write $\langle \mathcal{P}_\mathfrak{I}, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\rightarrow}} \langle A, \sqsubseteq\rangle$ when α is onto (or equivalently γ is injective or equivalently $\alpha \circ \gamma = \mathbb{1}_A$ is the identity).

In case of existence of a best abstraction, $\gamma \circ \alpha$ is an upper closure operator (increasing, extensive and idempotent) characterizing the abstraction (up to isomorphic representations A of the abstract domain $\gamma \circ \alpha(\mathcal{P}_\mathfrak{I})$). If $\langle \mathcal{P}_\mathfrak{I}, \subseteq\rangle$ is a complete lattice then so is its image $\langle \gamma \circ \alpha(\mathcal{P}_\mathfrak{I}), \subseteq\rangle$ by the an upper closure operator $\gamma \circ \alpha$ [Ward 1942, Th. 4.1], [Monteiro and Ribeiro 1942, Th. 8.2]. Moreover, all possible best abstractions are, up to concretization, given by the complete lattice of upper closure operators ordered pointwise on the complete lattice $\langle \mathcal{P}_\mathfrak{I}, \subseteq\rangle$ [Ward 1942, Th. 4.2], [Cousot and Cousot 1979a, Th. 4.3] (a result extended to CPOs by [Ranzato 1999]).

Given a concrete transformer $F_\mathcal{I}[\![\mathtt{P}]\!] \in \mathcal{P}_\mathcal{I} \to \mathcal{P}_\mathcal{I}$ the best abstract transformer is $\overline{F}_\mathcal{I}[\![\mathtt{P}]\!] \triangleq \alpha \circ F_\mathcal{I}[\![\mathtt{P}]\!] \circ \gamma$ which yields $\langle \mathcal{P}_\mathfrak{I} \xrightarrow{\nearrow} \mathcal{P}_\mathfrak{I},\ \dot{\subseteq}\rangle \underset{\lambda F \bullet \alpha \circ F \circ \gamma}{\overset{\lambda \overline{F} \bullet \gamma \circ \overline{F} \circ \alpha}{\leftrightarrows\!\!\!\!\rightarrow}} \langle A \xrightarrow{\nearrow} A,\ \dot{\sqsubseteq}\rangle$. In practice, the best transformer may be difficult to compute algorithmically, so that a strict over-approximation, such as $\overline{F}_\mathcal{I}[\![\mathtt{P}]\!] \mathrel{\dot{\sqsupseteq}} \alpha \circ F_\mathcal{I}[\![\mathtt{P}]\!] \circ \gamma$, has to be used instead.

[13] Note that in theorem 4.8, only condition $\forall y \in A : y \wedge y \leqslant x \mathbin{\triangledown} y$ is needed.

5. ABSTRACTION OF MULTI-INTERPRETED CONCRETE SEMANTICS

The interpreted concrete semantics of section **3.2** is relative to one interpretation $\mathfrak{I}$ of the programming language data, functions, and predicates. But the theories used in theorem provers or SMT solvers can have many different models, corresponding to possible interpretations. In fact, the same holds for programs: they can be executed on different platforms, and it can be useful to collect all the possible behaviors, e.g. to provide a more general proof of correctness (e.g. valid for all implementations according to the considered interpretations). In this case, the multi-interpreted concrete semantics of section **3.3** is useful.

5.1. Abstractions Between Multi-Interpretations

If we can only compute properties under one interpretation $\mathfrak{I}$, as in the case of section **3.2**, 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 abstracting in the sense that we add more behaviors and forget the actual property that should be associated with the program. So, in general, we have two sets of interpretations, one $\mathcal{I}$ is the context of interpretations for the program and the other $\mathcal{I}^\sharp$ is the set of interpretations used in the analysis. The relation between the two is a Galois connection.

LEMMA 5.1. *$\langle \mathcal{P}_\mathcal{I}, \subseteq\rangle \underset{\alpha_{\mathcal{I}\to\mathcal{I}^\sharp}}{\overset{\gamma_{\mathcal{I}^\sharp\to\mathcal{I}}}{\leftrightarrows}} \langle \mathcal{P}_{\mathcal{I}^\sharp}, \subseteq\rangle$ is a Galois connection where*

$$\alpha_{\mathcal{I}\to\mathcal{I}^\sharp}(P) \triangleq P \cap \mathcal{P}_{\mathcal{I}^\sharp}$$
$$\gamma_{\mathcal{I}^\sharp\to\mathcal{I}}(Q) \triangleq \left\{ \langle I, \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I, \eta\rangle \in Q\right) \right\} \qquad \square$$

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

$$\begin{array}{lll}
& \alpha_{\mathcal{I}\to\mathcal{I}^\sharp}(P) \subseteq Q & \\
\Leftrightarrow & P \cap \mathcal{P}_{\mathcal{I}^\sharp} \subseteq Q & \wr\text{def. } \alpha_{\mathcal{I}\to\mathcal{I}^\sharp}\wr \\
\Leftrightarrow & \forall \langle I, \eta\rangle \in P \cap \mathcal{P}_{\mathcal{I}^\sharp} : \langle I, \eta\rangle \in Q & \wr\text{def. } \subseteq\wr \\
\Leftrightarrow & \forall \langle I, \eta\rangle \in P : \langle I, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I, \eta\rangle \in Q & \wr\text{def. } \cap\wr \\
\Leftrightarrow & \forall \langle I, \eta\rangle \in P, I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge (\langle I, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I, \eta\rangle \in Q) & \\
& & \wr P \in \mathcal{P}_\mathcal{I} \simeq \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\}\wr \\
\Leftrightarrow & P \subseteq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge (\langle I, \eta\rangle \in \mathcal{P}_{\mathcal{I}^\sharp} \Rightarrow \langle I, \eta\rangle \in Q)\} & \wr\text{def. } \subseteq\wr \\
\Leftrightarrow & P \subseteq \left\{ \langle I, \eta\rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I, \eta\rangle \in Q\right) \right\} & \wr\mathcal{P}_{\mathcal{I}^\sharp} \simeq \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I}^\sharp \wedge \eta \in \mathcal{R}_I\}\wr \\
\Leftrightarrow & P \subseteq \gamma_{\mathcal{I}^\sharp\to\mathcal{I}}(Q) & \wr\text{def. } \gamma_{\mathcal{I}^\sharp\to\mathcal{I}}\wr \quad \square
\end{array}$$

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

Example 5.2. Considering the soundness of transformers defined in section **4.4** for the forward assignment of section **3.3**, we get, for all $P^\sharp \in \mathcal{P}_{\mathcal{I}^\sharp}$,

$$\begin{array}{ll}
& \mathrm{f}_\mathcal{I}[\![\mathtt{x} := e]\!] \circ \gamma_{\mathcal{I}^\sharp\to\mathcal{I}}(P^\sharp) \\
= & \left\{ \langle I, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \;\middle|\; \langle I, \eta\rangle \in \gamma_{\mathcal{I}^\sharp\to\mathcal{I}}(P^\sharp) \right\} \\
& \qquad \wr\text{def. } \mathrm{f}_\mathcal{I}[\![\mathtt{x} := e]\!]P \triangleq \left\{ \langle I, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \;\middle|\; \langle I, \eta\rangle \in P) \right\}\wr
\end{array}$$

$$= \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta \rangle \in P^\sharp \right) \right\} \qquad \text{(def. } \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \text{)}$$

$$= \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow (I \in \mathcal{I}^\sharp \wedge \langle I,\, \eta \rangle \in P^\sharp) \right) \right\} \qquad \text{(def. } \Rightarrow \text{)}$$

$$\subseteq \left\{ \langle I,\, \eta' \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta' \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta' \rangle \in \left\{ \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \;\middle|\; I \in \mathcal{I}^\sharp \wedge \eta \in \mathcal{R}_I \wedge \langle I,\, \eta \rangle \in P^\sharp \right\} \right) \right\}$$

(def. $\subseteq$)

$$= \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I \wedge \left(I \in \mathcal{I}^\sharp \Rightarrow \langle I,\, \eta \rangle \in \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp) \right) \right\}$$

(by defining $\mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!] \in \mathcal{P}_{\mathcal{I}^\sharp} \stackrel{\cdot}{\to} \mathcal{P}_{\mathcal{I}^\sharp}$ such that

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

)

$$\subseteq \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \circ \mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp) \qquad \text{(def. } \gamma_{\mathcal{I}^\sharp \to \mathcal{I}} \text{)}$$

Observe that $\mathsf{f}_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!]$ and $\mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!]$ have exactly the same definition. However, the corresponding post-fixpoint semantics do differ when $\mathcal{I}^\sharp \neq \mathcal{I}$ since $\langle \mathcal{P}_{\mathcal{I}^\sharp}, \subseteq \rangle \neq \langle \mathcal{P}_{\mathcal{I}}, \subseteq \rangle$. By changing the order on the lattice of properties one changes the least fixpoint/post-fixpoint abstract semantics. ∎

5.2. Homogeneous Abstraction of Interpretations

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

Example 5.3. That is what the Astrée analyzer [Cousot et al. 2005] does when taking all possible rounding error modes for floating points computations. ∎

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

$$\alpha_{\mathcal{I}}(P) \triangleq \left\{ \eta \;\middle|\; \exists I \in \mathcal{I} :\, \langle I,\, \eta \rangle \in P \right\} \quad \text{and} \quad \gamma_{\mathcal{I}}(E) \triangleq \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in E \right\}. \tag{10}$$

Proof. We have a Galois connection since for all $P \in \mathcal{P}_{\mathcal{I}}$ and $E \in \cup_{I \in \mathcal{I}} \mathcal{R}_I$,

$$\begin{array}{lll}
& \alpha_{\mathcal{I}}(P) \subseteq E & \\
\Leftrightarrow & \left\{ \eta \;\middle|\; \exists I \in \mathcal{I} :\, \langle I,\, \eta \rangle \in P \right\} \subseteq E & \text{(def. } \alpha_{\mathcal{I}}(P) \text{)} \\
\Leftrightarrow & \forall \eta : (\exists I \in \mathcal{I} :\, \langle I,\, \eta \rangle \in P) \Rightarrow \eta \in E & \text{(def. } \subseteq \text{)} \\
\Leftrightarrow & \forall \eta : \forall I \in \mathcal{I} :\, \langle I,\, \eta \rangle \in P \Rightarrow \eta \in E & \text{(def. } \Rightarrow \text{)} \\
\Leftrightarrow & \forall \langle I,\, \eta \rangle \in P : I \in \mathcal{I} \wedge \eta \in E & \text{(since } P \in \mathcal{P}_{\mathcal{I}} \simeq \wp(\{\langle I,\, \eta \rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\}) \text{)} \\
\Leftrightarrow & P \subseteq \left\{ \langle I,\, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge \eta \in E \right\} & \text{(def. } \subseteq \text{)} \\
\Leftrightarrow & P \subseteq \gamma_{\mathcal{I}}(E) & \text{(def. } \gamma_{\mathcal{I}}(E) \text{)} \quad \square
\end{array}$$

5.3. Abstraction by a Theory

In some cases it can be difficult to represent exactly an infinite set $\mathcal{I}$ of interpretations as proposed in section **5.1**. A solution is to use theories (preferably deductive with a recursively enumerable number of axioms) to represent the set $\mathcal{I} = \mathfrak{M}(\mathcal{T})$ of interpretations that are models of these theories. The relationship between theories and multi-interpreted semantics is expressed by the concretization function:

$$\gamma_{\mathfrak{M}}(\mathcal{T}) \triangleq \left\{ \langle I, \eta\rangle \,\middle|\, I \in \mathfrak{M}(\mathcal{T}) \right\} \tag{11}$$

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

Example 5.4. If $\mathfrak{I}$ interprets programs over the natural numbers $\mathbb{N}$, then by Gödel's first incompleteness theorem there is no enumerable first-order theory characterizing this interpretation, so the poset has no best abstraction of $\{\mathfrak{I}\}$. ∎

Once an (arbitrary) theory $\mathcal{T}$ has been chosen to abstract a set $\mathcal{I}$ of interpretations there is a best abstraction $\alpha_{\mathcal{I}\to\gamma_{\mathfrak{M}}(\mathcal{T})}(P) \triangleq P \cap \gamma_{\mathfrak{M}}(\mathcal{T})$ of interpreted properties in $P \in \mathcal{P}_{\mathcal{I}}$ by abstract properties in $\mathcal{P}_{\gamma_{\mathfrak{M}}(\mathcal{T})}$. By lemma 5.1, $\langle \mathcal{P}_{\mathcal{I}}, \subseteq\rangle \underset{\alpha_{\mathcal{I}\to\gamma_{\mathfrak{M}}(\mathcal{T})}(P)}{\overset{\gamma_{\gamma_{\mathfrak{M}}(\mathcal{T})\to\mathcal{I}}(P)}{\longleftarrow\!\!\!\!\!\!\longrightarrow}} \langle \mathcal{P}_{\gamma_{\mathfrak{M}}(\mathcal{T})}, \subseteq\rangle$, so the best abstract transformer is $\overline{F}_{\mathfrak{M}(\mathcal{T})}[\![\mathrm{P}]\!] \triangleq \alpha_{\mathcal{I}\to\gamma_{\mathfrak{M}}(\mathcal{T})} \circ F_{\mathcal{I}}[\![\mathrm{P}]\!] \circ \alpha_{\gamma_{\mathfrak{M}}(\mathcal{T})\to\mathcal{I}}$. However, there might be no finite formula to encode these best abstraction and best abstract transformer.

5.4. Algebraic Abstraction of Interpretations

Another direction for abstraction is to keep the context of interpretations and forget variable properties. This is simply a projection on the first component of the pairs of interpretation and environment. Given a set $\mathcal{I}$ of interpretations and, for each interpretation $I \in \mathcal{I}$, an algebraic abstraction $\langle \wp(\mathcal{R}_I), \subseteq\rangle \underset{\alpha_I}{\overset{\gamma_I}{\longleftarrow\!\!\!\!\longrightarrow}} \langle A_I, \sqsubseteq_I\rangle$, we have an abstraction

$$\langle \mathcal{P}_{\mathcal{I}}, \subseteq\rangle \underset{\dot\alpha}{\overset{\dot\gamma}{\longleftarrow\!\!\!\!\longrightarrow}} \langle \bigtimes_{I\in\mathcal{I}} A_I, \dot\sqsubseteq\rangle \tag{12}$$

of $\mathcal{P}_{\mathcal{I}} \simeq \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \mathcal{R}_I\})$ by defining

$$\begin{aligned} \dot\alpha(P) &\triangleq \bigtimes_{I\in\mathcal{I}} \alpha_I(\{\eta \mid \langle I, \eta\rangle \in P\}),\\ \dot\gamma(\overline{P}) &\triangleq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \gamma_I(\overline{P}_I)\},\\ \text{and}\quad \overline{P} \mathrel{\dot\sqsubseteq} \overline{Q} &\triangleq \forall I \in \mathcal{I} : \overline{P}_I \sqsubseteq_I \overline{Q}_I. \end{aligned}$$

Proof of (12). We have a Galois connection since for all $P \in \mathcal{P}_{\mathcal{I}}$ and $\overline{P} \in \bigtimes_{I\in\mathcal{I}} A_I$,

$$\begin{aligned} & \dot\alpha(P) \mathrel{\dot\sqsubseteq} \overline{P} \\ \Leftrightarrow\ & \bigtimes_{I\in\mathcal{I}} \alpha_I(\{\eta \mid \langle I, \eta\rangle \in P\}) \mathrel{\dot\sqsubseteq} \overline{P} && \text{⟅def. } \dot\alpha\text{⟆}\\ \Leftrightarrow\ & \forall I \in \mathcal{I} : \alpha_I(\{\eta \mid \langle I, \eta\rangle \in P\}) \sqsubseteq \overline{P}_I && \text{⟅pointwise def. of } \dot\sqsubseteq\text{⟆}\\ \Leftrightarrow\ & \forall I \in \mathcal{I} : \{\eta \mid \langle I, \eta\rangle \in P\} \subseteq \gamma_I(\overline{P}_I) && \text{⟅Galois connection } \langle \wp(\mathcal{R}_I), \subseteq\rangle \underset{\alpha_I}{\overset{\gamma_I}{\longleftarrow\!\!\!\!\longrightarrow}} \langle A_I, \sqsubseteq_I\rangle\text{⟆}\\ \Leftrightarrow\ & \forall I \in \mathcal{I} : \forall \eta : \langle I, \eta\rangle \in P \Rightarrow \eta \in \gamma_I(\overline{P}_I) && \text{⟅def. } \subseteq\text{⟆}\\ \Leftrightarrow\ & \forall \langle I, \eta\rangle \in P : I \in \mathcal{I} \wedge \eta \in \gamma_I(\overline{P}_I) && \text{⟅since } P \in \mathcal{P}_{\mathcal{I}} \simeq \wp(\{\langle I, \eta\rangle \mid I \in \mathcal{I} \Rightarrow \eta \in \mathcal{R}_I\}\text{⟆}\\ \Leftrightarrow\ & P \subseteq \{\langle I, \eta\rangle \mid I \in \mathcal{I} \wedge \eta \in \gamma_I(\overline{P}_I)\} && \text{⟅def. } \subseteq\text{⟆}\\ \Leftrightarrow\ & P \subseteq \dot\gamma(\overline{P}) && \text{⟅def. } \dot\gamma\text{⟆}\quad \square \end{aligned}$$

Of course if $\mathcal{I}$ is infinite, one may have to group interpretations in a finite partition, each block being abstracted uniformly e.g. as proposed in section **5.2**.

5.5. Comparative Abstraction of Interpretations

Another example of abstraction of multi-interpreted semantics consists in comparing interpretations such as the difference between a mathematical interpretation of programs on reals and an interpre-

tation on floats, to study the propagation of rounding errors in floating-point computations as in FLUCTUAT [Goubault et al. 2002].

We can also change the interpretation of expressions in section **2.3**, so as to keep track of all possible evaluations when applying mathematical identities such as commutativity, associativity, distributivity, *etc*. which hold for reals but not for floats. Again the abstraction keeps track for each order of evaluation of the difference when evaluating with reals and floats so as to determine the most precise evaluation order minimizing the rounding errors in arithmetic expressions [Martel 2009].

6. FIRST ORDER LOGICAL SEMANTICS

For theorem-prover based program verification, the multi-interpreted semantics of section **3.3** must be expressed using first-order logical formulæ. This involves an abstraction since, on one hand, not all concrete program properties and property transformers can be exactly expressed with logical formulæ and, on the other hand, not all concrete set-theoretic inclusions can be proved by logical implication.

6.1. Multi-Interpretation of First-Order Logic Formulæ

A logical formula $\Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ describes a property $\gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi)$ for multi-interpretations $\mathcal{I} \in \wp(\mathfrak{I})$ as follows

$$\begin{array}{rcl} \gamma^{\mathfrak{a}}_{\mathcal{I}} & \in & \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \xrightarrow{\cdot} \mathcal{P}_{\mathcal{I}} \\ \gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi) & \triangleq & \{\langle I,\ \eta\rangle \mid I \in \mathcal{I} \wedge I \models_{\eta} \Psi\} \end{array} \tag{13}$$

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

Example 6.1 (*Universal interpretation*). The universal interpretation consists in describing the properties encoded by a formula on all possible interpretations. Thus, the concretization of a formula will be given by $\gamma^{\mathfrak{a}}_{\mathfrak{I}}(\Psi) = \{\langle I,\ \eta\rangle \mid I \in \mathfrak{I} \wedge I \models_{\eta} \Psi\}$. ∎

Example 6.2 (*Strict approximation*). The use of first-order logic instead of set theory may enforce strict approximations in program verification [Cook 1978]. For example, in the context of Presburger arithmetic with interpretation $\mathbb{N}$ on the natural numbers, a program may compute a multiplication by successive additions, in which case the concrete set-theoretic property involving a multiplication (such as $P = \{\langle \mathbb{N},\ \eta\rangle \mid \eta(\mathtt{x}) = \eta(\mathtt{y}) \times \eta(\mathtt{z})\}$) may not be expressible by a finite first-order logical formula involving only addition. ∎

6.2. Axiomatic Semantics Modulo a Multi-Interpretation

Once concrete program properties have been abstracted by first-order logic formulæ, the concrete program semantics of section **3** must be abstracted in terms of first-order logic. Because this sound abstraction step will not lead to an effectively computable analysis, we aim for the most precise semantics at this point, and, in fact, we can usually be as precise as the concrete semantics.

As shown in [Cousot 2002] for safety properties, an axiomatic semantics $C_{\mathfrak{a}}[\![\mathtt{P}]\!]$ of a program P specifies program properties encoded by formulæ in $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ pre-ordered by implication $\Rrightarrow$. The axiomatic semantics $C_{\mathfrak{a}}[\![\mathtt{P}]\!] \in \wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}))$ is specified as a set of post-fixpoints:

$$C_{\mathfrak{a}}[\![\mathtt{P}]\!] \triangleq \left\{ \Psi \;\middle|\; F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \Rrightarrow \Psi \right\}$$

where $F_{\mathfrak{a}}[\![\mathtt{P}]\!] \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \xrightarrow{\cdot} \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ is the predicate transformer defining the axiomatic semantics of program P where the verification condition for $I \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ to be an inductive invariant of program P is $F_{\mathfrak{a}}[\![\mathtt{P}]\!](I) \Rrightarrow I$.

Observe that $(\Psi \Rrightarrow \Psi') \triangleq \mathsf{valid}(\Psi \Rightarrow \Psi')$ is a pre-order so that $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ is considered to be quotiented by $(\Psi \Lleftarrow\Rrightarrow \Psi') \triangleq \mathsf{valid}(\Psi \Leftrightarrow \Psi')$ meaning that logical formulæ are understood up to

equivalence. The resulting quotient poset[14] is a lattice $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}), \mapsto, \mathtt{ff}, \mathtt{tt}, \vee, \wedge \rangle$ although not a complete lattice (since infinite disjunctions or conjunctions are missing in first-order logic).

Again the program transformer $F_{\mathfrak{a}}$ can be defined in terms of primitive operations $\mathtt{ff}, \mathtt{tt}, \vee, \wedge, \nabla, \wedge, \mathsf{f}_{\mathfrak{a}}, \mathsf{b}_{\mathfrak{a}}, \mathsf{p}_{\mathfrak{a}}, \ldots$ whose local soundness conditions imply the soundness of the program transformer and the verification condition.

Example 6.3. Continuing example *3.2* in the context of invariance properties for imperative languages, the primitive operations will be `false`, `true`, $\vee$ for control flow joins, and the following primitives for assignments and tests:

$$\begin{array}{rcll} \mathsf{f}_{\mathfrak{a}} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \text{axiomatic forward assignment transformer} \\ \mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi & \triangleq & \exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x'] & \text{[15]} \\ \mathsf{b}_{\mathfrak{a}} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \text{axiomatic backward assignment transformer} \\ \mathsf{b}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi & \triangleq & \Psi[\mathtt{x} \leftarrow t] & \\ \mathsf{p}_{\mathfrak{a}} & \in & \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \to \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) & \text{axiomatic transformer for program test of condition } \varphi. \\ \mathsf{p}_{\mathfrak{a}}[\![\varphi]\!]\Psi & \triangleq & \Psi \wedge \varphi & \end{array}$$

■

6.3. Soundness of the Axiomatic Semantics Modulo a Multi-Interpretation

In general, the soundness of transformers

$$\forall \Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) : F_{\mathcal{I}}[\![\mathtt{P}]\!] \circ \gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi) \subseteq \gamma^{\mathfrak{a}}_{\mathcal{I}} \circ F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi), \tag{14}$$

follows from similar local soundness conditions on the operations of the abstract domain[16] (see e.g. example *4.5*). This implies that the axiomatic semantics modulo a multi-interpretation is sound.

THEOREM 6.4. *The axiomatic semantics $C_{\mathfrak{a}}[\![P]\!]$ of a program P is sound according to (8) with respect to a multi-interpreted semantics $C_{\mathcal{I}}[\![P]\!]$ of section **3.3** and the increasing concretization $\gamma^{\mathfrak{a}}_{\mathcal{I}}$ defined in (13).* □

PROOF. By theorem 4.4 using the instance (14) of (9). □

Example 6.5. Note that in the case of the axiomatic invariance semantics for imperative languages of example *6.3*, the interpretation of the axiomatic semantics is exactly the multi-interpreted concrete semantics. For example, for assignment,

$$\begin{array}{rcll} & \gamma^{\mathfrak{a}}_{\mathcal{I}}(\mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi) & & \\ \triangleq & \gamma^{\mathfrak{a}}_{\mathcal{I}}(\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x']) & & \text{(def. } \mathsf{f}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi\text{)} \\ = & \{\langle I, \eta \rangle \mid I \in \mathcal{I} \wedge I \models_{\eta} (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x'])\} & & \text{(def. (13) of } \gamma^{\mathfrak{a}}_{\mathcal{I}}\text{)} \\ = & \{\langle I, \eta'[\mathtt{x} \leftarrow [\![t]\!]_I \eta'] \rangle \mid I \in \mathcal{I} \wedge I \models_{\eta'} \Psi\} & & \\ & & \text{(since } I \models_{\eta} (\exists x' : \Psi[\mathtt{x} \leftarrow x'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow x']) \text{ if and only if } \exists \eta' : I \models_{\eta'} \Psi \text{ and } \eta = \eta'[\mathtt{x} \leftarrow [\![t]\!]_I \eta'] \text{ as defined in section } \mathbf{2.3}\text{)} & \\ = & \{\langle I, \eta[\mathtt{x} \leftarrow [\![t]\!]_I \eta] \rangle \mid I \in \mathcal{I} \wedge \langle I, \eta \rangle \in \{\langle I, \eta \rangle \mid I \models_{\eta} \Psi\}\} & & \text{(renaming } \eta' \text{ to } \eta \text{ and def. } \in\text{)} \\ = & \{\langle I, \eta[\mathtt{x} \leftarrow [\![t]\!]_I \eta] \rangle \mid I \in \mathcal{I} \wedge \langle I, \eta \rangle \in \gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi)\} & & \text{(def. (13) of } \gamma^{\mathfrak{a}}_{\mathcal{I}}\text{)} \\ = & \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := t]\!] \circ \gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi) & & \text{(def. (7) of } \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := t]\!]P\text{)} \end{array}$$

■

Example 6.6 (*Uninterpreted axiomatic semantics*). The uninterpreted axiomatic semantics corresponds to the universal interpretation of example *6.1*, that is $\mathcal{I} = \mathfrak{I}$. ■

---

[14] Following the tradition, we write $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}), \mapsto \rangle$ instead of the more rigorous quotient notation $\langle \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})/_{\Longleftrightarrow}, \mapsto \rangle$.

[15] $\Psi[\mathtt{x} \leftarrow t]$ is the substitution of term t for variable $\mathtt{x}$ in formula Ψ, assuming renaming whenever t has variables bound in Ψ.

[16] In general, we do not have equality since there may be no logical formula that exactly encodes $F_{\mathcal{I}}[\![\mathtt{P}]\!] \circ \gamma^{\mathfrak{a}}_{\mathcal{I}}(\Psi)$.

Example 6.7 (*Axiomatic semantics modulo theory*). An axiomatic semantics modulo a theory $\mathcal{T}$ corresponds to the choice $\mathcal{I} = \mathfrak{M}(\mathcal{T})$ so that the difference with the uninterpreted axiomatic semantics of example *6.6* is uniquely in the way implication can be proved with or without using a theorem of the theory $\mathcal{T}$. ∎

7. LOGICAL ABSTRACT DOMAINS

As a generalization of section **6**, we define logical abstract domains to perform program verification in the first-order logic setting. Computing the predicate transformer $\overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!]$ is quite immediate. The two hard points are

(1) the computation of the least fixpoint (or a post-fixpoint approximation of it since, in general, the logical lattice is not complete), and
(2) proving that the final formula implies the desired property.

To solve the first problem, the usual approach consists of asking the end-user to provide a solution or in restricting the set of formulæ used to represent program properties such that the ascending chain condition (ACC) is enforced. Using an infinite abstract domain not satisfying the ACC together with a widening can be much more precise [Cousot and Cousot 1992b].

Solving the second problem requires the proof of an implication for which a decidable theory can be used.

7.1. Definition of Logical Abstract Domains

We define logical abstract domains in the following general setting (without the ACC restriction):

Definition 7.1. *A* logical abstract domain *is* $\langle A, \sqsubseteq, \mathtt{ff}, \mathtt{tt}, \vee, \wedge, \nabla, \Delta, \bar{\mathtt{f}}_{\mathfrak{a}}, \bar{\mathtt{b}}_{\mathfrak{a}}, \bar{\mathtt{p}}_{\mathfrak{a}}, \ldots\rangle$ *defined by a pair* $\langle A,\ \mathcal{T}\rangle$ *of a set* $A \in \wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}))$ *of logical formulæ and of a theory* $\mathcal{T}$ *of* $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$. *The abstract properties* $\Psi \in A$ *define the concrete properties* $\gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) \triangleq \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_{\eta} \Psi \right\}$ *relative to the models* $\mathfrak{M}(\mathcal{T})$ *of theory* $\mathcal{T}$. *The abstract pre-order* $\sqsubseteq$ *on the abstract domain* $\langle A,\ \sqsubseteq\rangle$ *is defined as* $(\Psi \sqsubseteq \Psi') \triangleq ((\forall \vec{\mathbb{x}}_{\Psi} \cup \vec{\mathbb{x}}_{\Psi'} : \Psi \Rightarrow \Psi') \in \mathcal{T})$ *(and can be quotiented to a partial order by* $(\Psi \equiv \Psi') \triangleq ((\forall \vec{\mathbb{x}}_{\Psi} \cup \vec{\mathbb{x}}_{\Psi'} : \Psi \Leftrightarrow \Psi') \in \mathcal{T})$*).* ∎

This definition of logical abstract domains is close to the logical abstract interpretation framework developed by Gulwani and Tiwari [Gulwani and Tiwari 2006; Gulwani et al. 2008]. The main difference in our approach is that we consider a concrete semantics corresponding to the actual behavior of the program, whereas in the work of Gulwani and Tiwari, the behavior of the program is assumed to be described by formulæ in the same theory as the theory of the logical abstract domain, which may yield unsoundness. Our approach allows the description of the abstraction mechanism, the comparison of logical abstract domains, and the formal, rigorous proof of soundness.

7.2. Abstraction to Logical Abstract Domains

Because $A \in \wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}))$, we need to approximate formulæ in $\wp(\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})) \setminus A$ by a formula in A. The alternatives [Cousot and Cousot 1992a] are either to choose a context-dependent abstraction (a different abstraction is chosen in different circumstances, which can be understood as a widening extrapolation [Cousot 1978, Ch. 4.1]) or to define an abstraction function to use a uniform context-independent approximation whenever needed. The abstraction

$$\boldsymbol{\alpha}^{\mathcal{I}}_{A} \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}) \rightarrow A \qquad \text{abstraction (function/algorithm)}$$

abstracts a concrete first-order logic formula appearing in the axiomatic semantics into a formula in the logical abstract domain A. To be sound, the following must hold

$$\forall \Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p}),\ \forall I \in \mathcal{I} :\ I \models (\Psi \Rightarrow \boldsymbol{\alpha}^{\mathcal{I}}_{A}(\Psi)) \qquad \text{soundness} \tag{15}$$

The abstraction $\boldsymbol{\alpha}^{\mathcal{I}}_{A}$ can be chosen to be computable in which case we speak of an *abstraction algorithm*, which can be directly used in the implementation of the abstract domain and semantics.

When the abstraction $\alpha_A^{\mathcal{I}}$ is not computable, we speak of an *abstraction specification*, which has to be eliminated from the definition of the abstract domain and semantics (e.g. by automatic or manual design of an over-approximation of the abstract operations).

Example 7.2 (*Literal elimination*). Assume that the axiomatic semantics is defined on $\mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ and that the logical abstract domain is $A = \mathbb{F}(\mathbb{x}, \mathbb{f}_A, \mathbb{p}_A)$ where $\mathbb{f}_A \subseteq \mathbb{f}$ and $\mathbb{p}_A \subseteq \mathbb{p}$. The abstraction $\alpha_A^{\mathcal{I}}(\Psi)$ of $\Psi \in \mathbb{F}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ can be defined by repeating the following approximations until stabilization.

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

In both cases, this implies soundness (15) and the abstraction algorithm terminates since Ψ is finite. ∎

Example 7.3 (*Quantifier elimination*). If the abstract domain $A \subseteq \mathbb{C}(\mathbb{x}, \mathbb{f}_A, \mathbb{p}_A)$ is quantifier-free then the quantifiers must be eliminated, which is possible without loss of precision in some theories such as Presburger arithmetic (but with a potential blow-up of the formula size see e.g. [Cooper 1972; Ferrante and Rackoff 1975; Ferrante and Geiser 1977]). Otherwise, besides trivial simplifications of formulæ (e.g. replacing $\exists x : x = t \wedge \Psi[x]$ by $\Psi[t]$), a very coarse abstraction to $A \subseteq \mathbb{C}(\mathbb{x}, \mathbb{f}, \mathbb{p})$ would eliminate quantifiers bottom up, putting the formula in disjunctive normal form and eliminating the literals containing existentially quantified variables (or dually [McMillan 2002]), again with a potential blow-up. Other proposals of abstraction functions (often not identified as such) include the quantifier elimination heuristics defined in Simplify [Detlefs et al. 2005, Sect. 5], [de Moura et al. 2003, Sect. 6], or the (doubly-exponential) methods of [Ge et al. 2007; Ge and de Moura 2009] (which might even be made more efficient when exploiting the fact that an implication rather than an equivalence is required). ∎

Example 7.4 (*Interval abstraction*). Let us consider the minimal abstraction α_m, whose reduced product with the maximal abstraction α_M, yields the interval abstraction [Cousot and Cousot 1976; Cousot and Cousot 1977].

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

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

7.3. Abstract Logical Transformers

For soundness with respect to a set of interpretations $\mathcal{I}$, the abstract transformers must be chosen such that [Cousot and Cousot 1977; Cousot and Cousot 1979c] for each program $\mathtt{P}$

$$\begin{aligned} \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!] &\in A \to A && \text{abstract transformer} \\ \forall \Psi \in A, \forall I \in \mathcal{I} : I \models F_{\mathfrak{a}}[\![\mathtt{P}]\!]\Psi &\Rightarrow I \models \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!]\Psi && \text{abstract transformer soundness} \end{aligned} \tag{16}$$

Again, the abstract program transformer $\overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!]$ can be defined in terms of primitive operations (e.g. $\overline{\mathsf{f}}_{\mathfrak{a}}$, $\overline{\mathsf{b}}_{\mathfrak{a}}$, and $\overline{\mathsf{p}}_{\mathfrak{a}}$ of example *4.1*) satisfying local soundness conditions, which imply the soundness

of the program transformer. In many static analyzers, the abstract transfer functions are usually designed, proved correct and implemented by hand.

Example 7.5. Continuing example *6.3*, the abstract logical transformers should be designed so as to satisfy the following local soundness conditions

$$\begin{array}{rcll}
\bar{\mathsf{f}}_\mathfrak{a} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to A \to A & \text{abstract forward assignment transformer} \\
\forall \Psi \in A, \forall I \in \mathcal{I} :: I \models \mathsf{f}[\![\mathtt{x} := t]\!]\Psi & \Rightarrow & \bar{\mathsf{f}}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi & \text{abstract postcondition soundness} \\
\overline{\mathsf{b}}_\mathfrak{a} & \in & (\mathbb{x} \times \mathbb{T}(\mathbb{x}, \mathbb{f})) \to A \to A & \text{abstract backward assignment transformer} \\
\forall \Psi \in A, \forall I \in \mathcal{I} :: I \models \mathsf{b}[\![\mathtt{x} := t]\!]\Psi & \Rightarrow & \overline{\mathsf{b}}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi & \text{abstract precondition soundness} \\
\overline{\mathsf{p}}_\mathfrak{a} & \in & \mathbb{L} \to A \to A & \text{condition abstract transformer} \\
\forall \Psi \in A, \forall I \in \mathcal{I} : \mathsf{p}[\![l]\!]\Psi & \Rightarrow & \overline{\mathsf{p}}_\mathfrak{a}[\![l]\!]\Psi & \text{abstract test soundness}
\end{array}$$

It follows from the definition of the uninterpreted axiomatic semantics in example *6.6* that we can define the abstract transformer to be the axiomatic transformer. This requires a closure hypothesis on A to ensure that they map a formula in A to a formula in A. Otherwise, an overapproximation may be necessary. For example, A may just contain formulæ without disjunction $\vee$ so that disjunction must be overapproximated. This is one of the uses of widening [Cousot 1978] (the other being to enforce convergence of iterates as in section **4.5**).

$$\forall \Psi_1, \Psi_2 \in A, \forall I \in \mathcal{I} : I \models (\Psi_1 \vee \Psi_2 \Rightarrow \Psi_1 \mathbin{\overline{\nabla}_\mathfrak{a}} \Psi_2) \qquad \text{widening soundness}$$ ∎

This design and implementation should be totally automatized, resorting to a manual solution only when automation is too inefficient or imprecise. For example, when an abstraction algorithm $\boldsymbol{\alpha}_A^{\mathcal{I}}$ is available (section **7.2**), a simple sound implementation of the abstract transformers would be

$$\overline{F}_\mathfrak{a}[\![\mathsf{P}]\!]\Psi \triangleq \boldsymbol{\alpha}_A^{\mathcal{I}}(F_\mathfrak{a}[\![\mathsf{P}]\!]\Psi) .$$

Example 7.6. Continuing example *6.3* for an abstraction $\boldsymbol{\alpha}_A^{\mathcal{I}}$ of section **7.2**, the abstract logical transformers would be

$$\begin{array}{rcll}
\bar{\mathsf{f}}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi & \triangleq & \boldsymbol{\alpha}_A^{\mathcal{I}}(\mathsf{f}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi) & \text{abstract forward assignment transformer} \\
\overline{\mathsf{b}}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi & \triangleq & \boldsymbol{\alpha}_A^{\mathcal{I}}(\mathsf{b}_\mathfrak{a}[\![\mathtt{x} := t]\!]\Psi) & \text{abstract backward assignment transformer} \\
\overline{\mathsf{p}}_\mathfrak{a}[\![\varphi]\!]\Psi & \triangleq & \boldsymbol{\alpha}_A^{\mathcal{I}}(\mathsf{p}_\mathfrak{a}[\![\varphi]\!]\Psi) & \text{abstract transformer for program test of condition } \varphi \\
\Psi_1 \mathbin{\overline{\nabla}_\mathfrak{a}} \Psi_2 & \triangleq & \boldsymbol{\alpha}_A^{\mathcal{I}}(\Psi_1 \vee \Psi_2) & \text{abstract lub widening.}
\end{array}$$ ∎

These abstract transformers (or some over-approximations satisfying local soundness conditions as in example *7.5*) might be automatically computable using solvers (see e.g. [Reps et al. 2004] when A satisfies the ACC).

Example 7.7. With the interval abstraction of example *7.4*, where the abstract domain is $A = \{\bigwedge_{\mathtt{x} \in \mathbb{x}} \mathsf{c}_\mathtt{x} \leqslant \mathtt{x} \mid \forall \mathtt{x} \in \mathbb{x} : \mathsf{c}_\mathtt{x} \in \mathbb{f}^0\}$ and $\boldsymbol{\alpha}_A^{\mathcal{I}} = \boldsymbol{\alpha}_m$, an SMT solver (e.g. with linear arithmetic or even simple inequalities [Pratt 1977]) might be usable when restricting Ψ in $\boldsymbol{\alpha}_m(\Psi)$ to the formulæ obtained by the transformation of formulæ of A by the abstract transformers of example *7.6*. ∎

Finally the logical abstract transformer can be defined using a specific local abstraction.

Example 7.8 (*Abstract assignment*). The non-invertible assignment transformer returns a quantified formula

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

which may have to be abstracted to A can be realized using the abstraction α of section **7.2** or the widening of section **4.5** or on the fly, using program specificities. For example, in straight line code outside of iteration or recursion, the existential quantifier can be eliminated

— using logical equivalence, by Skolemization where $\forall x_1 : \ldots \forall x_n : \exists y : p(x_1, \ldots, x_n, y)$ is replaced by the equi-satisfiable formula $\forall x_1 : \ldots \forall x_n : p(x_1, \ldots, x_n, f_y(x_1, \ldots, x_n))$ where f_y is a fresh symbol function;
— using a program transformation, since x' denotes the value of the variable $\mathtt{x}$ before the assignment we can use a program equivalence introducing new fresh program variable $\mathtt{x}'$ to store this value since "$\mathtt{x} := t$" is equivalent to "$\mathtt{x}' := \mathtt{x};\ \mathtt{x} := t[\mathtt{x} \leftarrow \mathtt{x}']$" [17]. We get

$$\bar{\mathsf{f}}_{\mathfrak{a}}[\![\mathtt{x} := t]\!]\Psi \triangleq \Psi[\mathtt{x} \leftarrow \mathtt{x}'] \wedge \mathtt{x} = t[\mathtt{x} \leftarrow \mathtt{x}'] \quad \text{abstract non-invertible assignment}$$

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

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

7.4. Soundness of the Abstract Logical Semantics

The abstract logical semantics $\overline{C}_{\mathfrak{a}}[\![\mathtt{P}]\!] \in \wp(A)$ of a program $\mathtt{P}$ in a logical abstract domain $\langle A, \sqsubseteq\rangle$ defined by $\langle A, \mathcal{T}\rangle$ is specified in post-fixpoint form (since, in general, least fixpoints do not exist).

$$\overline{C}_{\mathfrak{a}}[\![\mathtt{P}]\!] \triangleq \left\{ \Psi \in A \;\middle|\; \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \Rrightarrow \Psi \right\}$$

Theorem 7.9 (Soundness of the abstract logical semantics). *Under the abstract transformer soundness hypotheses (14) with $\mathcal{I} = \mathfrak{M}(\mathcal{T})$ and (16), the abstract logical semantics $\overline{C}_{\mathfrak{a}}[\![P]\!]$ of a program P is sound according to (8) with respect to a multi-interpreted semantics $C_{\mathfrak{M}(\mathcal{T})}[\![P]\!]$ of section **3.3** and the increasing concretization $\gamma^{\mathfrak{a}}_{\mathcal{T}} \triangleq \gamma^{\mathfrak{a}}_{\mathfrak{M}(\mathcal{T})}$ where (13) is relative to the models $\mathfrak{M}(\mathcal{T})$ of theory $\mathcal{T}$.* □

Proof. By (13), $\gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) \triangleq \gamma^{\mathfrak{a}}_{\mathfrak{M}(\mathcal{T})}(\Psi) = \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta \Psi \right\}$. First, we know that $\gamma^{\mathfrak{a}}_{\mathcal{T}}$ is increasing for the implication in $\mathcal{T}$. Then it is also increasing for $\sqsubseteq$, as $(\Psi \sqsubseteq \Psi') \triangleq ((\forall \vec{\mathtt{x}}_\Psi \cup \vec{\mathtt{x}}_{\Psi'} : \Psi \Rightarrow \Psi') \in \mathcal{T})$, and this implies that $\mathcal{T} \models \Psi \Rightarrow \Psi'$. Then we prove (9), that is for all $\Psi \in A$,

$$\begin{array}{lll}
& \gamma^{\mathfrak{a}}_{\mathcal{T}} \circ \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) & \\
= & \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \right\} & \text{\textwon since } \gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) = \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta \Psi \right\} \text{\textwoff} \\
\supseteq & \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) \right\} & \text{\textwon def. } \subseteq \text{ and (16) so that } I \models F_{\mathfrak{a}}[\![\mathtt{P}]\!]\Psi \text{ implies } I \models \overline{F}_{\mathfrak{a}}[\![\mathtt{P}]\!]\Psi \text{\textwoff} \\
= & \gamma^{\mathfrak{a}}_{\mathcal{T}} \circ F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) & \text{\textwon since } \gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) = \left\{ \langle I,\ \eta\rangle \;\middle|\; I \in \mathfrak{M}(\mathcal{T}) \wedge I \models_\eta \Psi \right\} \text{\textwoff} \\
= & \gamma^{\mathfrak{a}}_{\mathfrak{M}(\mathcal{T})} \circ F_{\mathfrak{a}}[\![\mathtt{P}]\!](\Psi) & \text{\textwon def. } \gamma^{\mathfrak{a}}_{\mathcal{T}} \text{\textwoff} \\
\supseteq & F_{\mathfrak{M}(\mathcal{T})}[\![\mathtt{P}]\!] \circ \gamma^{\mathfrak{a}}_{\mathfrak{M}(\mathcal{T})}(\Psi) & \text{\textwon hypothesis (14) with } \mathcal{I} = \mathfrak{M}(\mathcal{T}) \text{\textwoff} \\
= & F_{\mathfrak{M}(\mathcal{T})}[\![\mathtt{P}]\!] \circ \gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) & \text{\textwon def. } \gamma^{\mathfrak{a}}_{\mathcal{T}}(\Psi) \text{\textwoff}
\end{array}$$

Theorem 7.9 then immediately follows from theorem 4.4. □

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

7.5. Approximations of the Abstract Ordering

When implementing a logical abstract domain, in addition to the transfer functions, we need to implement the abstract order $\sqsubseteq$ by providing an effective decision procedure for formulæ of the form $\forall \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \Rightarrow \Psi'$. For efficiency reasons, or because the theory is undecidable, one may approximate this decision procedure. Then the approximation must be correct with respect to the positive answer to the decision problem: if the approximate algorithm answers that the formula is in the theory, then that must hold.

In the particular case of decision procedures based on SMT solvers, we can only decide satisfiability results. So we will rephrase our formulæ into $\exists \vec{\mathbf{x}}_\Psi \cup \vec{\mathbf{x}}_{\Psi'} : \Psi \wedge \neg\Psi'$. In this case, the answer of an SMT solver must be correct on the negative answer: if the solver says that the formula is unsatisfiable, then it must be the case to be unsatisfiable in the theory. So, for our applications, the refutation completeness of SMT solvers is not compulsory to insure soundness.

Notice that in section **7.4**, the soundness of the abstract logical semantics, in particular theorem 7.9, is with respect to the abstract order $\sqsubseteq$ and theory $\mathcal{T}$ of definition 7.1, with respect to the concrete order $\Rightarrow$. It follows that if the approximation of the abstract order $\sqsubseteq$ changes (e.g. due to a modification in or change of the theorem prover or SMT solver), then the soundness proof of the static analysis remains valid. The limit of the iterates with widening in theorem 4.8 also remains sound with respect to the concrete semantics for sound modifications of the approximation of the abstract order. A change in the approximation of the abstract ordering may change the cost and precision of the analysis (since provable facts may no longer be provable or vice-versa) but its result, although possibly different, always remains sound.

This would not be necessarily the case if the order $\sqsubseteq$ of definition 7.1 had been chosen to refer to the approximation algorithm, hence to a specific version of a theorem prover or SMT solver, since a change in the definition of the abstract order would have required to redo all soundness proofs.

7.6. Logical Widening and Narrowing

Designing a universal widening for logical abstract domains is difficult since powerful widenings prevent infinite evolution in the semantic computation, evolution that does not always manifest itself as a syntactic evolution in logical abstract domains. Nevertheless, we can propose several possible widenings.

(1) Widen to a finite sub-domain W of A organized in a partial order choosing $X \mathbin{\triangledown} Y$ to be $\Psi \in W$ such that $Y \Rightarrow \Psi$ and starting from the smallest elements of W (or use a further abstraction into W as in section **7.2**);
(2) Limit the size of formulæ to $k > 0$, eliminating new literals in the simple conjunctive normal form appearing beyond the fixed maximal size (e.g. depth) k (the above widenings are always sound and terminating but not very satisfactory, see [Cousot and Cousot 1992b]);
(3) Follow the syntactic evolution of successive formulæ and reduce the evolving parts as proposed by [Mauborgne 1998] for Typed Decision Graphs.
(4) Make generalizations (e.g. $l(1) \vee l(2) \vee \ldots$ implies $\exists k \geqslant 0 : l(k)$ and abstract the existential quantifier, see example *7.3*) or use saturation [18] [Ganzinger 1996].
(5) Use a *bounded widening* $\triangledown(u)$ when an upper bound u is known (e.g. from specifications to be checked) so that $x \mathbin{\triangledown(u)} y$ where $x \sqsubseteq y$ satisfies either $x \sqsubseteq y \sqsubseteq x \mathbin{\triangledown(u)} y \sqsubseteq u$ when $y \sqsubseteq u$ or else $x \mathbin{\triangledown(u)} y = \top$ (since specification u cannot be proved to be satisfied anyway). Notice that the widening may not use at all its first argument x, in which case a *dual narrowing* satisfying $y \sqsubseteq y \mathbin{\tilde{\triangle}} u \sqsubseteq u$ whenever $y \sqsubseteq u$ can also be used (provided convergence is enforced by widening to the upper bound u after a number of non-convergent steps and beyond to $\top$ if necessary). Craig interpolation [Craig 1957] is an example for logical abstract domains [McMillan 2003].

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

7.7. Enforcing Soundness of Unsound Abstractions

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

Still, such reasoning can give some informations about the program provided the invariant they find is precise enough. One way for them to be correct for an interpretation $\mathfrak{I}$ is to have one model of the theory to agree with $\mathfrak{I}$ on the formulæ that appear during the computation. Formally, two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ agree on Ψ when $\left\{ \eta \;\middle|\; I_1 \models_\eta \Psi \right\} = \left\{ \eta \;\middle|\; I_2 \models_\eta \Psi \right\}$

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

In section **12.2**, after studying the reduced product between logical abstract domains and algebraic abstract domains, we will show how this early detection can be extended so that even when such unsound cases occur, the analysis can proceed.

8. OBSERVATIONAL SEMANTICS

Program static analysis involves the of the possible data manipulated by programs such as values of variables, the control stack or the heap. It may also be interesting to observe other quantities such as functions of program control and data. An example is the inclusion of auxiliary variables in the Owicki and Gries proof method of parallel programs to observe the control of other processes (since assertions cannot directly refer to program counters as in Lamport's method) [Cousot 1990]. Another example is the queue of threads associated with wait conditions in monitors [Hoare 1974], which cannot be directly manipulated by programs but to which correctness proofs must refer. Similarly for programming languages for which Hoare logic is incomplete [Cousot 1990], the assertions might have to refer to the local variables of procedures passed as parameters whose values may have to be tracked in the control stack although they may not be directly visible at some program point. The idea of observing the norm of quaternions in the analysis of space programs [Bertrane et al. 2010] is similar. Finally, some data may have to be decomposed into pieces observed separately so that the data are a function of these pieces. This is the case in [Chen et al. 2011] where a vector $x = (x_i)_{i=1}^n$ is observed as $x^+ \triangleq (\max(x_i, 0))_{i=1}^n$ and $x^- \triangleq (\max(-x_i, 0))_{i=1}^n$ so that $x = x^+ - x^-$ and $|x| = x^+ + x^-$.

It is possible to cope with all these cases in a uniform way by explicitly defining the observables of the program semantics

8.1. Observable Properties of Multi-interpreted Programs

We name observables by identifiers in $\mathbb{x}_{\mathcal{O}}$ (which, in particular, can be variable identifiers in $\mathbb{x}_{\mathrm{P}}$).

$\Sigma = \langle \mathbb{x}, \mathbb{f}, \mathbb{p}, \# \rangle$	signature
$\Sigma_{\mathrm{P}} = \langle \mathbb{x}_{\mathrm{P}}, \mathbb{f}, \mathbb{p}, \# \rangle \dot{\subseteq} \Sigma$	program signature ($\mathbb{x}_{\mathrm{P}} \subseteq \mathbb{x}$)
$\mathtt{x} \in \mathbb{x}_{\mathrm{P}}$	program variables
$\Sigma_{\mathcal{O}} = \langle \mathbb{x}_{\mathcal{O}}, \mathbb{f}, \mathbb{p}, \# \rangle \dot{\subseteq} \Sigma$	observable signature ($\mathbb{x}_{\mathcal{O}} \subseteq \mathbb{x}$)
$x \in \mathbb{x}_{\mathcal{O}}$	observable identifiers

Program properties are sets of environments relative to interpretations $I \in \mathfrak{I}(\Sigma)$.

$$\begin{array}{rcll} v & \in & I_{\mathcal{V}} & \text{values (for interpretation } I \in \mathfrak{I}(\Sigma)) \\ \eta & \in & \mathcal{R}_I^{\Sigma_{\mathrm{P}}} \triangleq \mathbb{x}_{\mathrm{P}} \to I_{\mathcal{V}} & \text{program variable environments} \\ \boldsymbol{I} & \in & \wp(\mathfrak{I}(\Sigma)) & \text{multiple interpretations} \\ \mathfrak{R}_{\boldsymbol{I}}^{\Sigma_{\mathrm{P}}} & \triangleq & \left\{ \langle I,\ \eta\rangle \,\middle|\, I \in \boldsymbol{I} \wedge \eta \in \mathcal{R}_I^{\Sigma_{\mathrm{P}}} \right\} & \text{multi-interpreted program environment} \\ \mathfrak{P}_{\boldsymbol{I}}^{\Sigma_{\mathrm{P}}} & \triangleq & \wp(\mathfrak{R}_{\boldsymbol{I}}^{\Sigma_{\mathrm{P}}}) & \text{multi-interpreted program properties} \end{array}$$

Observables are functions from values of program variables to values that are named by observable identifiers.

$$\begin{array}{rcll} \omega_I & \in & O_I^{\Sigma_{\mathrm{P}}} \triangleq \mathcal{R}_I^{\Sigma_{\mathrm{P}}} \to I_{\mathcal{V}} & \text{observables (for } I \in \boldsymbol{I}) \\ \Omega_I & \in & \mathbb{x}_O \to O_I^{\Sigma_{\mathrm{P}}} & \text{observable naming.} \end{array}$$

The same way that the semantics keeps tracks of values of variables in program variable environments, the semantics keeps tracks of values of observables in program observable environments.

$$\begin{array}{rcll} \zeta & \in & \mathcal{R}_I^{\Sigma_O} \triangleq \mathbb{x}_O \to I_{\mathcal{V}} & \text{program observable environments} \\ \mathfrak{R}_{\boldsymbol{I}}^{\Sigma_O} & \triangleq & \left\{ \langle I,\ \zeta\rangle \,\middle|\, I \in \boldsymbol{I} \wedge \zeta \in \mathcal{R}_I^{\Sigma_O} \right\} & \text{multi-interpreted environments} \\ \mathfrak{P}_{\boldsymbol{I}}^{\Sigma_O} & \triangleq & \wp(\mathfrak{R}_{\boldsymbol{I}}^{\Sigma_O}) & \text{multi-interpreted observable properties} \end{array}$$

Whereas a concrete *program semantics* is relative to $\mathfrak{P}_{\boldsymbol{I}}^{\Sigma_{\mathrm{P}}}$, the *observational semantics* is relative to $\mathfrak{P}_{\boldsymbol{I}}^{\Sigma_O}$ and both can be specified in fixpoint or in post-fixpoint form. The difference is that the value $\eta(\mathtt{x})$ of a program variable $\mathtt{x} \in \mathbb{x}_{\mathrm{P}}$ in a variable environment $\eta \in \mathcal{R}_I^{\Sigma_{\mathrm{P}}}$ is modified when this variable $\mathtt{x}$ is modified whereas the value $\zeta(x)$ of an observable variable $x \in \mathbb{x}_O$ in an observable environment $\zeta \in \mathcal{R}_I^{\Sigma_O}$ is modified whenever the value $\Omega_I(x)\eta$ of the function $\Omega_I(x)$ observed under the observable name x is modified due to the modification of the variable environment $\eta \in \mathcal{R}_I^{\Sigma_{\mathrm{P}}}$ following the modification of the value of some program variable(s).

Example 8.1 (*Variable multiple observations*). We may want to observe variable values at different time instants, as abstracted to program points; this leads to SSA [Cytron et al. 1991] renaming variables such that the property of static single assignment holds. ∎

Example 8.2 (*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 $x \in \mathbb{x}_O \setminus \mathbb{x}_{\mathrm{P}}$ so that $\Omega_I(x) \triangleq [\![t]\!]_I$. For example, quaternions are analyzed in [Bertrane et al. 2010] by observing the value of their norm $t = \sqrt{\mathtt{a}^2 + \mathtt{b}^2 + \mathtt{c}^2 + \mathtt{d}^2}$ (where $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d} \in \mathbb{x}_{\mathrm{P}} \cup \mathbb{x}_O$ are variables) for several rounding semantics of floats. ∎

Example 8.3 (*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 e.g. [Deutsch 1990]. ∎

Example 8.4 (*Memory model*). In the memory model of [Miné 2006a], 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, $\eta(\mathtt{x}) = \Omega_I(x_3)\eta \times 2^{24} + \Omega_I(x_2)\eta \times 2^{16} + \Omega_I(x_1)\eta \times 2^8 + \Omega_I(x_0)\eta$. ∎

Given a program property $P \in \mathfrak{P}_{\boldsymbol{I}}^{\Sigma_{\mathrm{P}}}$, the corresponding observable property is

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

The value of the observable named x is therefore $\Omega_I(x)\eta$ where the values of program variables are given by η. Conversely, given an observable property $Q \in \mathfrak{P}_I^{\Sigma_O}$, the corresponding program property is

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

Example 8.5 (*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 $\boldsymbol{\lambda}\, \mathtt{x} \bullet \Omega_I(\mathtt{x})$, $\boldsymbol{\alpha}_I^{\Omega}$ and $\boldsymbol{\gamma}_I^{\Omega}$ are the identity. If we do not want to analyze the values of all program variables then $\mathbb{x}_O \subsetneq \mathbb{x}_P$. ∎

The observation of the value of functions of the variables is less precise that the direct observation of these variables so we have a Galois connection between the program and observable properties.

THEOREM 8.6. $\langle \mathfrak{P}_I^{\Sigma_P},\, \subseteq\rangle \underset{\boldsymbol{\alpha}_I^{\Omega}}{\overset{\boldsymbol{\gamma}_I^{\Omega}}{\leftrightarrows}} \langle \mathfrak{P}_I^{\Sigma_O},\, \subseteq\rangle$. □

PROOF.

$$\begin{array}{lll}
& \boldsymbol{\alpha}_I^{\Omega}(P) \subseteq Q & \\
\Leftrightarrow & \left\{ \langle I,\, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta\rangle \in \mathfrak{R}_I^{\Sigma_O} \;\middle|\; \langle I,\, \eta\rangle \in P \right\} \subseteq Q & \text{⟅def. } \boldsymbol{\alpha}_I^{\Omega}\text{⟆} \\
\Leftrightarrow & \forall \langle I,\, \eta\rangle \in P : \langle I,\, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta\rangle \in Q & \text{⟅def. } \subseteq \text{ and } Q \in \mathfrak{P}_I^{\Sigma_O} = \wp(\mathfrak{R}_I^{\Sigma_O})\text{⟆} \\
\Leftrightarrow & P \subseteq \left\{ \langle I,\, \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \;\middle|\; \langle I,\, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta\rangle \in Q \right\} & \text{⟅def. } \subseteq \text{ and } P \subseteq \mathfrak{R}_I^{\Sigma_P}\text{⟆} \\
& P \subseteq \boldsymbol{\gamma}_I^{\Omega}(Q) & \text{⟅def. } \boldsymbol{\gamma}_I^{\Omega}\text{.⟆}
\end{array}$$ □

Nevertheless, there is no loss of precision whenever the observables include all program variables. In this case, there is also no gain of precision in the concrete. However the gain of precision may be quite significative in the abstract whenever the abstraction is tailored to the observation functions.

8.2. Soundness of the Abstraction of Observable Properties

The *observational abstraction* is an abstraction 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 $\mathfrak{P}_I^{\Sigma_P}$ is the case where $\mathbb{x}_O = \mathbb{x}_P$ and $\boldsymbol{\lambda}\, \mathtt{x} \bullet \Omega_I(\mathtt{x})$ is identity. The program properties corresponding to observable Ω_I are given by $\boldsymbol{\gamma}_I^{\Omega,P} \in A_I^{\Sigma_O} \mapsto \mathfrak{P}_I^{\Sigma_P}$ such that

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

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.

Example 8.7. The soundness condition of the assignment abstract postcondition $\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]$ becomes:

Lemma 8.8. $\boldsymbol{\gamma}_I^{\Omega,P}(\bar{\mathtt{f}}\,[\![\mathtt{x} := e]\!]\overline{P}) \;\supseteq\; \mathtt{f}_I[\![\mathtt{x} := e]\!](\boldsymbol{\gamma}_I^{\Omega,P}(\overline{P}))$ and similarly for the other transformers.

PROOF.

$$\begin{array}{lll}
& \boldsymbol{\gamma}_I^{\Omega,P}(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}) \;\supseteq\; \mathtt{f}_I[\![\mathtt{x} := e]\!](\boldsymbol{\gamma}_I^{\Omega,P}(\overline{P})) & \\
\Leftrightarrow & \left\{ \langle I,\, \eta\rangle \in \mathfrak{R}_I^{\Sigma_P} \;\middle|\; \langle I,\, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta\rangle \in \gamma_I^{\Sigma_O}(\bar{\mathtt{f}}[\![\mathtt{x} := e]\!]\overline{P}) \right\} \;\supseteq & \\
\end{array}$$

$$\left\{ \langle I,\ \eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta] \rangle \;\middle|\; \langle I,\ \eta \rangle \in \Re_{\mathcal{I}}^{\Sigma_{\mathrm{P}}} \wedge \langle I,\ \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta \rangle \in \gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}(\overline{P}) \right\}$$

$$\text{(def. } \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}, \mathsf{f}_{\mathcal{I}}[\![\mathtt{x} := e]\!], \text{ and } \boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}\text{)}$$

$$\Leftrightarrow \quad \forall \langle I,\ \eta \rangle \in \Re_{\mathcal{I}}^{\Sigma_{\mathrm{P}}} : \forall \langle I,\ \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta \rangle \in \gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}(\overline{P}) :$$
$$\langle I,\ \boldsymbol{\lambda}\, x \bullet \Omega_I(x)(\eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]) \rangle \in \gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}\left(\bar{\mathsf{f}}[\![\mathtt{x} := e]\!]\overline{P}\right) \qquad \text{(def. } \subseteq\text{)}$$

$$\Leftrightarrow \quad \gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}\left(\bar{\mathsf{f}}[\![\mathtt{x} := e]\!]\overline{P}\right) \supseteq \left\{ \langle I,\ \boldsymbol{\lambda}\, x \bullet \Omega_I(x)(\eta[\mathtt{x} \leftarrow [\![e]\!]_I \eta]) \rangle \;\middle|\; \langle I,\ \eta \rangle \in \Re_{\mathcal{I}}^{\Sigma_{\mathrm{P}}} \wedge \langle I,\ \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta \rangle \in \gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}(\overline{P}) \right\} \qquad \text{(def. } \subseteq\text{)} \quad \square \quad \blacksquare$$

8.3. Observational Extension

It can sometimes be useful to extend an abstract property $\overline{P}$ for observables Ω with a new observable ω named x. For example, this was useful for intervals in [Elder et al. 2010]. We will write $\mathsf{extend}_{(x,\omega)}\left(\overline{P}\right)$ for the extension of $\overline{P}$ with the observable ω for the observable identifier x.

Example 8.9. Let $A_{\mathbb{x}_{\mathcal{O}}}$ be the abstract domain mapping observable identifiers $\mathtt{x} \in \mathbb{x}_{\mathcal{O}}$ to an interval of values [Cousot and Cousot 1977]. Assume that intervals of program variables are observable, that is $\mathbb{x}_{\mathrm{P}} \subseteq \mathbb{x}_{\mathcal{O}}$, and let $\mathtt{x} \in \mathbb{x}_{\mathrm{P}}$ be a program variables for which we want to observe the square $\mathtt{x}^2$ so $\omega_I \triangleq [\![\mathtt{x}^2]\!]_I$. Let $\mathtt{x2} \notin \mathbb{x}_{\mathcal{O}}$ be a fresh name for this observable. This extension of observable properties with a new observable $\mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])} \in A_{\mathbb{x}_{\mathcal{O}}} \to A_{\mathbb{x}_{\mathcal{O}} \cup \{\mathtt{x2}\}}$ can be defined as

$$\mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])} \ \in \ A_{\mathbb{x}_{\mathcal{O}}} \to A_{\mathbb{x}_{\mathcal{O}} \cup \{\mathtt{x2}\}}$$
$$\mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])}\left(\overline{P}\right) \ \triangleq \ \boldsymbol{\lambda}\, x \in \mathbb{x}_{\mathcal{O}} \cup \{\mathtt{x2}\} \bullet (\ x \neq \mathtt{x2}\ ?\ \overline{P}(x)\ :\ \overline{P}(\mathtt{x2}) \otimes \overline{P}(\mathtt{x2})\ )$$

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

The semantics of this extension operation must satisfy the following soundness condition

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

The introduction of auxiliary variables to name alien terms in logical abstract domains is an observational extension of the domains.

Lemma 8.10. *For the logical abstract domain* $A \triangleq \mathbb{F}(\Sigma)$ *with* $\gamma_{\mathcal{I}}^{\Sigma_{\mathcal{O}}}(\Psi) \triangleq \left\{ \langle I, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge I \models_\eta \Psi \right\}$,

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

Proof.

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

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

Definition 8.11 (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 $\mathbb{x}_O \subseteq \mathbb{x}$, the set of interpretations $\mathcal{I}$, and observables named by Ω such that $\forall I \in \mathcal{I} : \Omega_I \in \mathbb{x}_O \to O_I^{\Sigma_P}$.*

Consider the new observables Ω' such that $\forall I \in \mathcal{I} : \Omega'_I \in (\mathbb{x}_{O'} \setminus \mathbb{x}_O) \to O_I^{\Sigma_P}$ 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*

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

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

9. REDUCED PRODUCT

9.1. Cartesian and Reduced Product

The Cartesian product can be used for the conjunction of static analyzes [Cousot and Cousot 1979c].

Definition 9.1 (Cartesian product). *Let $\langle A_i, \sqsubseteq_i \rangle$, $i \in \Delta$, Δ finite, be abstract domains with increasing concretization $\gamma_i \in A_i \xrightarrow{\nearrow} \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$. 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 $\vec{\gamma} \in \vec{A} \to \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ is $\vec{\gamma}(\vec{P}) \triangleq \bigcap_{i\in\Delta} \gamma_i(\vec{P}_i)$.* $\blacksquare$

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 $\gamma_{ij}(\langle x,\, y\rangle) \triangleq \gamma_i(x) \cap \gamma_j(y)$. Notice that instead of $\langle \mathfrak{P}_{\mathcal{I}}^{\Sigma_O},\, \subseteq,\, \emptyset,\, \mathfrak{R}_{\mathcal{I}}^{\Sigma_O},\, \cup,\, \cap \rangle$ where $\mathfrak{P}_{\mathcal{I}}^{\Sigma_O} \triangleq \wp(\mathfrak{R}_{\mathcal{I}}^{\Sigma_O})$, the concrete domain could also be chosen as an arbitrary poset $\langle L,\, \leqslant \rangle$, meet semi-lattice $\langle L,\, \leqslant,\, \wedge \rangle$, cpo $\langle L,\, \leqslant,\, 0,\, \vee \rangle$, or complete lattice $\langle L,\, \leqslant,\, 0,\, 1,\, \vee,\, \wedge \rangle$.

If abstract transformers are applied component-wise, then the Cartesian product of static analyzes yields exactly the same result as running analyses with each abstract domain independently and performing the conjunction of their respective results. To improve this result, the reduced product was defined [Cousot and Cousot 1979c] so that each analysis benefits from the information brought by the other analyses.

Definition 9.2 (Reduced product (I)). *Let $\langle A_i, \sqsubseteq_i \rangle$, $i \in \Delta$, Δ finite, be abstract domains with increasing concretization $\gamma_i \in A_i \xrightarrow{\nearrow} \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ where $\vec{A} \triangleq \bigtimes_{i\in\Delta} A_i$ is their Cartesian product. Their* reduced product *is $\langle \vec{A}/_{\vec{\equiv}}, \vec{\sqsubseteq} \rangle$ 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 $\vec{\equiv}$ by $\vec{\gamma}([\vec{P}]/_{\vec{\equiv}}) = \vec{\gamma}(\vec{P})$ and $[\vec{P}]/_{\vec{\equiv}} \vec{\sqsubseteq} [\vec{Q}]/_{\vec{\equiv}} \triangleq \exists \vec{P}' \in [\vec{P}]/_{\vec{\equiv}} : \exists \vec{Q}' \in [\vec{Q}]/_{\vec{\equiv}} : \vec{P}' \vec{\sqsubseteq} \vec{Q}'$.* $\blacksquare$

The reduced product can yield much more precise results than the Cartesian product by computing more precise abstract values for each abstract domain, while staying in the same class of the reduced product. Computing such abstract values is naturally a reduction (see section **9.3**) where information from one abstract domain is transferred to other abstract domains to increase their precision.

Example 9.3. A classical example [Cousot and Cousot 1979c] is the product of a sign and a parity analysis where the discovery that $\mathtt{x} = 0$ by the sign analysis and that $\mathtt{x} \bmod 2 = 1$ by the parity analysis in a test/guard yields $\bot$ (non-reachability) for both abstract domains, a fact that neither abstraction could infer by itself thus missing unreachability of subsequent code (which may

also be the case of their conjunction for this subsequent code). Sign and parity reduction [Cousot and Cousot 1979c] was generalized to intervals and simple congruences in [Granger 1989]. ∎

9.2. The Reduced Product is the Greatest Lower Bound in the Poset of Abstract Domains

We can compare the expressiveness of abstract domains by defining an abstract domain A_1 to be more precise than A_2 whenever any property exactly expressible by A_2 is also expressible by A_1. Two abstract domains are equivalent when they are equally expressive.

Definition 9.4 (Precision of abstractions). *Let $\langle A_i, \sqsubseteq_i\rangle$, $i \in \{1,2\}$, be abstract domains with concretization $\gamma_i \in A_i \to L$ into the concrete domain $\langle L, \leqslant\rangle$. We say that A_2 is* less precise (also *expressive*, *refined*, *etc*...) than A_1 *(written $A_1 \trianglelefteq A_2$) whenever $\gamma_2(A_2) \subseteq \gamma_1(A_1)$. They are* equivalent *whenever $\gamma_1(A_1) = \gamma_2(A_2)$ (written $A_1 \trianglelefteqslant A_2$).* ∎

So $\langle \wp(L), \supseteq\rangle$ is isomorphic to the complete lattice of all abstract domains quotiented by $\trianglelefteqslant$ and ordered by precision $\trianglelefteq$. Each abstract domain $\langle A, \sqsubseteq\rangle$ is $\trianglelefteqslant$-equivalent to an element of this lattice $\wp(L)$ of all abstract domains [Cousot and Cousot 1979c]. In case of abstractions $A_1 \triangleq \rho_1(L)$ and $A_2 \triangleq \rho_2(L)$ defined by upper closures ρ_1 and ρ_2 on $\langle L, \leqslant\rangle$, we have $A_1 \trianglelefteq A_2 \Leftrightarrow \rho_2(L) \subseteq \rho_1(L) \Leftrightarrow \rho_1 \dot{\leqslant} \rho_2$ [Cousot and Cousot 1979c]. In case of abstractions defined by Galois connections $\langle L, \leqslant\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle A_1, \sqsubseteq_1\rangle$ and $\langle L, \leqslant\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle A_2, \sqsubseteq_2\rangle$, we have $A_1 \trianglelefteq A_2 \Leftrightarrow \gamma_1 \circ \alpha_1(L) \dot{\leqslant} \gamma_2 \circ \alpha_2(L)$ [Cousot and Cousot 1979c].

Definition 9.5 (Closure by intersection). *An abstract domain $\langle A, \sqsubseteq\rangle$ with concretization $\gamma \in A \xrightarrow{\wedge} L$ into a meet semi-lattice (resp. complete lattice) $\langle L, \leqslant, \wedge\rangle$ is* closed by finite (resp. infinite) intersection *if and only if $\forall P, Q \in A : \exists R \in A : \gamma(R) = \gamma(P) \wedge \gamma(Q)$ (resp. $\forall \mathcal{P} \in \wp(A) : \exists R \in A : \gamma(R) = \bigwedge \gamma(\mathcal{P})$).* ∎

When considering only abstract domains that are closed by finite intersection, the reduced product can equivalently be understood as the greatest lower bound in the complete lattice of abstract domains, up to the equivalence $\trianglelefteqslant$.

Theorem 9.6 (Equivalent definition of the reduced product (II)). *Let the meet semi-lattice $\langle L, \leqslant, 1, \wedge\rangle$ be a concrete domain, $\langle A_i, \sqsubseteq_i, \top_i\rangle$, $i \in \Delta$, Δ finite, be abstract domains with increasing concretization $\gamma_i \in A_i \xrightarrow{\wedge} L$ and supremum $\top_i$ such that $\gamma_i(\top_i) = 1$. The reduced product $\langle \vec{A}/_{\equiv}, \vec{\sqsubseteq}, [\vec{\top}]/_{\equiv}\rangle$ is*

— *more precise than $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$,*
— *less precise than any other $\langle \overline{L}, \overline{\sqsubseteq}\rangle$, which is closed by finite intersection and more precise than $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$.*

If $\langle A_i, \sqsubseteq_i, \top_i\rangle$, $i \in \Delta$ are closed by finite intersection then their reduced product $\langle \vec{A}/_{\equiv}, \vec{\sqsubseteq}, [\vec{\top}]/_{\equiv}\rangle$ is the unique such abstract domain (up to the equivalence $\trianglelefteqslant$). □

Proof. — For all $i \in \Delta, P_i \in A_i$, we have $\vec{\gamma}([\vec{\top}]/_{\equiv}[i \leftarrow P_i]) \triangleq \bigwedge_{j\in\Delta\setminus\{i\}} \gamma_i(\top_j) \wedge \gamma_i(P_i) = \bigwedge_{j\in\Delta\setminus\{i\}} 1 \wedge \gamma_i(P_i) = 1 \wedge \gamma_i(P_i) = \gamma_i(P_i)$ proving that $\gamma_i(A_i) \subseteq \vec{\gamma}(\vec{A}/_{\equiv})$ so that $\langle \vec{A}/_{\equiv}, \vec{\sqsubseteq}\rangle$ is more precise than the $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$.
— If $\langle \overline{L}, \overline{\sqsubseteq}\rangle$ with $\overline{\gamma} \in \overline{L} \xrightarrow{\wedge} L$ is more precise than the $\langle A_i, \sqsubseteq_i\rangle$ then $\gamma_i(A_i) \subseteq \overline{\gamma}(\overline{L})$. Given $\vec{P} \in \vec{A}/_{\equiv}$, we have $\gamma_i(\vec{P}_i) \in \overline{\gamma}(\overline{L})$ so there exists $\overline{P} \in \overline{L}$ such that $\overline{\gamma}(\overline{P}) = \bigwedge_{i\in\Delta} \gamma_i(\vec{P}_i) \triangleq \vec{\gamma}(\vec{P})$ since $\langle \overline{L}, \overline{\sqsubseteq}\rangle$ is closed by finite intersection and Δ is finite. It follows that $\vec{\gamma}(\vec{A}/_{\equiv}) \subseteq \overline{\gamma}(\overline{L})$.
— The property is characteristic since the $\langle A_i, \sqsubseteq_i, \top_i\rangle$, $i \in \Delta$ are closed by finite intersection so that their reduced product $\langle \vec{A}/_{\equiv}, \vec{\sqsubseteq}, [\vec{\top}]/_{\equiv}\rangle$ is also closed by intersection and therefore any other abstract domain $\langle \overline{\overline{L}}, \overline{\overline{\sqsubseteq}}\rangle$ with the same property would have both $\vec{\gamma}(\vec{A}/_{\equiv}) \subseteq \overline{\overline{\gamma}}(\overline{\overline{L}})$ and $\overline{\overline{\gamma}}(\overline{\overline{L}}) \subseteq \vec{\gamma}(\vec{A}/_{\equiv})$ hence would, by antisymmetry, be an equivalent abstract domain. □

9.3. Abstract Domain Reduction

Different abstract domains $A_1 \leftrightarrows A_2$ with the same expressive power may yield different abstract properties which, after iteration, may yield sound but quite different results. In particular, this is the case when the best transformer is difficult to compute algorithmically, so that a strict over-approximation $\overline{F}_{\mathcal{I}}[\![\mathrm{P}]\!] \dot{\supset} \alpha \circ F_{\mathcal{I}}[\![\mathrm{P}]\!] \circ \gamma$ has to be used instead. In such a case, reduction may be useful.

Example 9.7. Consider the abstraction of $\langle \wp(\mathbb{Z}), \subseteq \rangle$ by the complete lattice $\langle A, \sqsubseteq \rangle$ where $A \triangleq \{\bot, 0, +, +1, -, -1, \top\}$, $\bot \sqsubset 0 \sqsubset + \sqsubset +1 \sqsubset \top$ and $0 \sqsubset - \sqsubset -1 \sqsubset \top$ with $\gamma(+) = \gamma(+1) = \{z \in \mathbb{Z} \mid z \geqslant 0\}$ and $\gamma(-) = \gamma(-1) = \{z \in \mathbb{Z} \mid z \leqslant 0\}$. The positive and negative properties have distinct but equivalent encodings in the abstract. The two transformers $f_1(0) = f_1(+) = +$, $f_1(+1) = \top$ and $f_2(0) = +$, $f_2(+) = +1$, $f_2(+1) = \top$ are equivalent in the concrete in that $\forall \overline{P} \in A : \gamma(f_1(\overline{P})) = \gamma(f_2(\overline{P}))$ but their composition is not since $\gamma(f_1(f_1(f_1(0)))) = \gamma(+) \neq \gamma(\top) = \gamma(f_2(f_2(f_2(0))))$. ∎

Another example is the reduction of the Cartesian product of abstract domains into their reduced product.

Example 9.8. Consider the Cartesian product of sign and parity in example *9.3* with a reduction of the abstract value of $\mathtt{x}$ from positive or zero and odd to strictly positive [Cousot and Cousot 1979c]. The transformer for $1/\mathtt{x}$ now yields positive instead of $\top$ as required when $\mathtt{x}$ can be zero, a strict improvement of precision. ∎

The reduction of an abstract domain consists in eliminating the redundant abstract properties by putting them in normal form.

Example 9.9. Continuing example *9.7*, let us define the reduction $\rho_\gamma(a) \triangleq \bigsqcap \left\{ a' \in A \;\middle|\; \gamma(a) \leqslant \gamma(a') \right\}$ such that $\rho_\gamma(+1) = +$, $\rho_\gamma(-1) = -$ and otherwise $\rho_\gamma(a) = a$. The reduced abstract domain is $\rho_\gamma(A) = \{\bot, 0, +, -, \top\}$ where the redundant abstract properties $+1$ and -1 have been eliminated. ∎

Theorem 9.10 (Reduction Operator). *Let $\langle C, \leqslant \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq \rangle$ where $\langle A, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is a complete lattice. Define*

$$\rho_\gamma(a) \triangleq \bigsqcap \left\{ a' \in A \;\middle|\; \gamma(a) \leqslant \gamma(a') \right\}$$

then ρ_γ is a lower closure (reductive, increasing and idempotent) and

$$\langle C, \leqslant \rangle \underset{\rho_\gamma \circ \alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} \langle \rho_\gamma(A), \sqsubseteq \rangle \,. \qquad \square$$

Proof. —— ρ_γ is reductive since $a \in \{a' \in A \mid \gamma(a) \leqslant \gamma(a')\}$ by reflexivity and so $\rho_\gamma(a) \leqslant a$ by def. glb $\sqcap$.

—— If $a \leqslant b$ then $\gamma(a) \leqslant \gamma(b)$ so $\gamma(b) \leqslant \gamma(b')$ implies $\gamma(a) \leqslant \gamma(b')$ hence $\{b' \mid \gamma(b) \leqslant \gamma(b')\} \subseteq \{a' \mid \gamma(a) \leqslant \gamma(a')\}$ so $\rho_\gamma(a) = \bigsqcap\{a' \mid \gamma(a) \leqslant \gamma(a')\} \leqslant \bigsqcap\{b' \mid \gamma(b) \leqslant \gamma(b')\} = \rho_\gamma(b)$.

—— For idempotence, we have

$$\begin{aligned}
&\rho_\gamma(\rho_\gamma(a)) \\
&= \textstyle\bigsqcap\{a' \in A \mid \gamma(\rho_\gamma(a)) \leqslant \gamma(a')\} && \text{(def. } \rho_\gamma\text{)} \\
&= \textstyle\bigsqcap\{a' \in A \mid \gamma(\bigsqcap\{a'' \in A \mid \gamma(a) \leqslant \gamma(a'')\}) \leqslant \gamma(a')\} && \text{(def. } \rho_\gamma\text{)} \\
&= \textstyle\bigsqcap\{a' \in A \mid \bigwedge\{\gamma(a'') \in A \mid \gamma(a) \leqslant \gamma(a'')\} \leqslant \gamma(a')\} && \text{(In a Galois connection, } \gamma \text{ preserves meets)} \\
&= \textstyle\bigsqcap\{a' \in A \mid \gamma(a) \leqslant \gamma(a')\} && \text{(since } \gamma(a) = \textstyle\bigwedge\{\gamma(a'') \in A \mid \gamma(a) \leqslant \gamma(a'')\} \text{ by reflexivity and def. glb)} \\
&= \rho_\gamma(a) && \text{(def. } \rho_\gamma\text{)}
\end{aligned}$$

— We conclude that ρ_γ is a lower closure (reductive, increasing and idempotent) on A.

— By the Galois connection, $x \leqslant \gamma(y)$ implies $\alpha(x) \leqslant y$ implies $\rho_\gamma \circ \alpha(x) \leqslant y$ since ρ_γ is a lower closure hence reductive and $y = \rho_\gamma(y)$ is closed.

— Inversely, if $x \in C$ and $y \in \rho_\gamma(A)$ then

$$\begin{array}{lll}
& \rho_\gamma \circ \alpha(x) \leqslant y & \\
\Rightarrow & \bigsqcap\{a' \in A \mid \gamma(\alpha(x)) \leqslant \gamma(a')\} \leqslant y & \wr\text{def. } \circ \text{ and } \rho_\gamma\wr \\
\Rightarrow & \gamma(\bigsqcap\{a' \in A \mid \gamma(\alpha(x)) \leqslant \gamma(a')\}) \leqslant \gamma(y) & \wr\text{In a Galois connection, } \gamma \text{ increasing}\wr \\
\Rightarrow & (\bigwedge\{\gamma(a') \in A \mid \gamma(\alpha(x)) \leqslant \gamma(a')\}) \leqslant \gamma(y) & \wr\text{In a Galois connection, } \gamma \text{ preserves existing glbs}\wr \\
\Rightarrow & \gamma \circ \alpha(x) \leqslant \gamma(y) & \wr\text{reflexivity for } a' = \alpha(x) \text{ and def. glb}\wr \\
\Rightarrow & x \leqslant \gamma(y) & \wr\text{In a Galois connection, } \gamma \circ \alpha \text{ is extensive and transitivity}\wr \quad \square
\end{array}$$

The following theorem shows that the reduction does not change the meaning of reduced abstract properties.

THEOREM 9.11. $\gamma = \gamma \circ \rho_\gamma$ □

PROOF. For all $x \in C$:

$$\begin{array}{lll}
& \gamma \circ \rho_\gamma \circ \alpha(x) & \\
= & \gamma(\dot{\bigsqcap}\{a \mid \gamma(\alpha(x)) \leqslant \gamma(a)\}) & \wr\text{def. } \rho_\gamma\wr \\
= & \bigwedge\{\gamma(a) \mid \gamma(\alpha(x)) \leqslant \gamma(a)\}) & \wr\text{In a Galois connection, } \gamma \text{ preserves meets}\wr \\
= & \gamma(\alpha(x)) & \wr\text{choosing } a = \alpha(x) \text{ and def. glb}\wr
\end{array}$$

and so

$$\begin{array}{lll}
& \gamma & \\
= & \gamma \circ \alpha \circ \gamma & \wr\text{Galois connection}\wr \\
= & \gamma \circ \rho_\gamma \circ \alpha \circ \gamma & \wr\text{since } \gamma \circ \alpha = \gamma \circ \rho_\gamma \circ \alpha\wr \\
\leqslant & \gamma \circ \rho_\gamma & \wr\alpha \circ \gamma \text{ is reductive and } \gamma \text{ and } \rho_\gamma \text{ are increasing}\wr
\end{array}$$

Moreover ρ_γ is a lower closure on $\langle \bigtimes_{i\in\Delta} A_i, \leqslant_\Delta \rangle$ so ρ_γ is reductive ($\rho_\gamma \mathbin{\dot{\sqsubseteq}_\Delta} \mathbb{1}$) hence $\gamma \circ \rho_\gamma \mathbin{\dot{\sqsubseteq}} \gamma$ since γ is increasing. By antisymmetry, $\gamma \circ \rho_\gamma = \gamma$. □

As shown by example *9.7*, although abstract domains and transformers may have equivalent concretizations, the equivalent but more precise abstract properties propagate through several transformers and the resulting transformed properties and fixpoint computations may be ultimately more precise in the concrete.

Notice also that an implementation of the reduction of an abstract domain A does not need considerable modifications of the implementation of A. The reduction operator ρ_γ may simply be applied before and after the operations of abstract domain A (although maybe not after a widening/narrowing since the reduction might introduce divergences).

9.4. The Reduced Product is the Meaning-Preserving Reduction of the Cartesian Product

The reduced product of abstract domains A_i, $i \in \Delta$ can be encoded by a reduction of the Cartesian product $\bigtimes_{i\in\Delta} A_i$ using a meaning-preserving reduction operator $\vec{\rho}$ mapping any element of the Cartesian product to the smallest representative of its equivalence class. This yields a more constructive definition of reduction and later leads to algorithms to perform or approximate this reduction.

Definition 9.12 (Meaning-preserving Map/Reduction). *Let $\langle A, \sqsubseteq\rangle$ be a poset that is an abstract domain with concretization $\gamma \in A \xrightarrow{\nearrow} C$ where $\langle C, \leqslant\rangle$ is the concrete domain. A* meaning-preserving map *is $\rho \in A \to A$ such that $\forall \overline{P} \in A : \gamma(\rho(\overline{P})) = \gamma(\overline{P})$. The map is a* reduction *if and only if $\forall \overline{P} \in A : \rho(\overline{P}) \sqsubseteq \overline{P}$.* ∎

Theorem 9.13 (Equivalent definition of the reduced product (III)). *Assume that $\langle L, \leqslant\rangle$ is a poset and $\forall i \in \Delta$, each $\langle A_i, \sqsubseteq_i, 0_i, 1_i, \sqcup_i, \sqcap_i\rangle$ is a complete lattice such that $\langle L, \leqslant\rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle A_i, \sqsubseteq_i\rangle$. Let $\langle \vec{A}, \vec{\sqsubseteq}\rangle$ be the Cartesian product of the $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$ with concretization $\vec{\gamma}$ as in definition 9.1. Define $\vec{\alpha} \triangleq \boldsymbol{\lambda}\, x \bullet \bigtimes_{i\in\Delta} \alpha_i(x)$ such that $\langle L, \leqslant\rangle \underset{\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows}} \langle \vec{A}, \vec{\sqsubseteq}\rangle$. Let $\vec{\rho} \triangleq \boldsymbol{\lambda}\, \vec{P} \bullet \bigwedge\{\vec{P'} \mid \vec{\gamma}(\vec{P}) \leqslant \vec{\gamma}(\vec{P'})\}$ such that $\langle L, \leqslant\rangle \underset{\vec{\rho}\circ\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} \langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$.*

Then $\vec{\rho}$ is meaning-preserving and $\langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ is the reduced product of the $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$. □

Proof. —— $\langle L, \leqslant\rangle \underset{\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows}} \langle \vec{A}, \vec{\sqsubseteq}\rangle$ and $\langle L, \leqslant\rangle \underset{\vec{\rho}\circ\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} \langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ follow directly from the hypothesis $\forall i \in \Delta : \langle L, \leqslant\rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle A_i, \sqsubseteq_i\rangle$.

—— Let us show that $\vec{\rho}$ is meaning-preserving.

$$
\begin{aligned}
& \vec{\gamma} \circ \vec{\rho}(\vec{P}) \\
=\ & \vec{\gamma}\Big(\bigwedge\{\vec{P'} \mid \vec{\gamma}(\vec{P}) \leqslant \vec{\gamma}(\vec{P'})\}\Big) && \text{⟅def. } \vec{\rho}\text{⟆} \\
=\ & \bigwedge\{\vec{\gamma}(\vec{P'}) \mid \vec{\gamma}(\vec{P}) \leqslant \vec{\gamma}(\vec{P'})\} && \text{⟅}\langle L, \leqslant\rangle \underset{\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows}} \langle \vec{A}, \vec{\sqsubseteq}\rangle \text{ so } \vec{\gamma} \text{ preserves existing glb⟆} \\
=\ & \vec{\gamma}(\vec{P}) && \text{⟅choosing } \vec{P'} = \vec{P} \text{ and def. glb⟆}
\end{aligned}
$$

—— It follows that $\langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ is more precise that the $\langle A_i, \sqsubseteq_i\rangle$ in that $\vec{\gamma} \circ \vec{\rho} \circ \vec{\alpha} \dot{\leqslant} \gamma_i \circ \alpha_i$ as follows.

$$
\begin{aligned}
& \vec{\gamma} \circ \vec{\rho} \circ \vec{\alpha}(x) \\
=\ & \vec{\gamma} \circ \vec{\alpha}(x) && \text{⟅}\vec{\rho} \text{ is meaning-preserving⟆} \\
=\ & \bigwedge_{k\in\Delta} \gamma_k\Big(\big(\bigtimes_{i\in\Delta} \alpha_i(x)\big)_k\Big) && \text{⟅def. } \vec{\gamma} \text{ and } \vec{\alpha}\text{⟆} \\
=\ & \bigwedge_{k\in\Delta} \gamma_k(\alpha_k(x)) && \text{⟅def. index selection⟆} \\
\vec{\sqsubseteq}\ & \gamma_i \circ \alpha_i(x) && \text{⟅for any } i \in \Delta\text{, by def. glb⟆}
\end{aligned}
$$

—— Let M be an abstraction $\langle L, \leqslant\rangle \underset{\vec{\alpha'}}{\overset{\vec{\gamma'}}{\leftrightarrows}} \langle M, \vec{\sqsubseteq}\rangle$ that is more precise than the $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$ in that $\forall i \in \Delta : \vec{\gamma'} \circ \vec{\alpha'} \dot{\vec{\sqsubseteq}} \gamma_i \circ \alpha_i$. So $\vec{\gamma'} \circ \vec{\alpha'} \dot{\vec{\sqsubseteq}} \dot{\bigwedge}_{i\in\Delta} \gamma_i \circ \alpha_i = \vec{\gamma} \circ (\vec{\rho} \circ \vec{\alpha})$ as just shown above, so $\langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ is less precise than $\langle M, \vec{\sqsubseteq}\rangle$.

—— In conclusion, for all $i \in \Delta$, $\langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle \trianglelefteqslant \langle A_i, \sqsubseteq_i\rangle$ and if $\forall i \in \Delta : \langle M, \vec{\sqsubseteq}\rangle \trianglelefteqslant \langle A_i, \sqsubseteq_i\rangle$ then $\langle M, \vec{\sqsubseteq}\rangle \trianglelefteqslant \langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ proving, by theorem 9.6, that $\langle \vec{\rho}(\vec{A}), \vec{\sqsubseteq}\rangle$ is the reduced product of $\langle A_i, \sqsubseteq_i\rangle$, $i \in \Delta$. □

10. ITERATED REDUCED PRODUCT

The strong reduction operators ρ_γ and $\vec{\rho}$ of sections **9.3** and **9.4** may be difficult to compute algorithmically. In that case one may use a weaker reduction, which makes a property more precise in the abstract without changing its concrete meaning. The precision of such a weak reduction can be improved in the abstract by iteration. This yields a weaker form of reduced product by iteration of pairwise weak reductions.

10.1. Iterated Weak Reduction

By iterating a weak reduction, one can improve even more the precision of a static analysis without altering its soundness. A particular case of iterated reduction was proposed by [Granger 1992] following [Cousot and Cousot 1979c].

Definition 10.1 (Iterated Reduction). *Let $\langle A, \sqsubseteq\rangle$ be a poset that is an abstract domain with concretization $\gamma \in A \xrightarrow{\nearrow} C$ where $\langle C, \subseteq\rangle$ is the concrete domain and $\rho \in A \to A$ be a meaning-preserving reduction.*

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 $\langle A, \sqsubseteq\rangle$.* ■

Theorem 10.2 (Finite iterated reduction). *The finite iterates ρ^n, $n \in \mathbb{N}$ of a meaning-preserving reduction ρ on $\langle A, \sqsubseteq\rangle$ are meaning-preserving and more precise in the abstract.* □

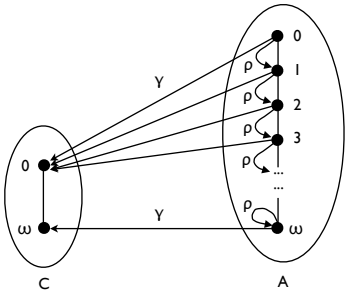
Proof. Let $\langle \rho^n,\ n \in \mathbb{N}\rangle$ be the iterates of a meaning-preserving reduction ρ. Observe, by recurrence, that the iterates form a descending chain since ρ is reductive so $\forall n < m : \rho^n(\overline{P}) \sqsubseteq \rho^n(\overline{P}) \sqsubseteq \overline{P}$. Meaning-preservation follows by recurrence. For the basis, $\gamma(\rho^0(\overline{P})) = \gamma(\overline{P})$ by def. of ρ^0. For induction, $\gamma(\rho^{n+1}(\overline{P})) \triangleq \gamma(\rho(\rho^n(\overline{P}))) = \gamma(\rho^n(\overline{P})) = \gamma(\overline{P})$, since ρ is meaning-preserving and by induction hypothesis. □

Notice, however, that the limit of the iterates of a meaning-preserving reduction may not be meaning-preserving.

Theorem 10.3 (Limit of an iterated reduction). *Let ρ be a meaning-preserving reduction that iterates from $\overline{P}$ are well-defined. Then their limit $\rho^*(\overline{P})$ exists. We have $\forall \beta < \lambda : \rho^*(\overline{P}) \sqsubseteq \rho^\lambda(\overline{P}) \sqsubseteq \rho^\beta(\overline{P}) \sqsubseteq \overline{P}$, but the limit is in general not meaning-preserving.* □

Proof. Assuming the iterates of ρ from $\overline{P} \in A$ are well-defined, we observe, by transfinite induction, that the iterates form a descending chain since ρ is reductive and by definition of the glb $\bigsqcap$. By the antisymmetry of $\sqsubseteq$ in the poset $\langle A, \sqsubseteq\rangle$, a fixpoint must be reached at rank ϵ of the iterates 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 ϵ 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.

To prove that the limit is in general unsound, consider the following example.



The finite iterates of the reduction ρ from 0 form a decreasing chain whose elements are all meaning-preserving since $\gamma(\rho^n(0)) = \gamma(0)$. These iterates have a well-defined limit $\rho^*(0) = \bigsqcap_{n\in\mathbb{N}} \rho^n(0) = \omega$ which is a reduction, but not meaning-preserving, since $\gamma(\omega) \neq \gamma(0)$. □

We now study sufficient conditions for the limit of a meaning-preserving reduction to itself be meaning-preserving. To do so we study how to improve a meaning-preserving map or to combine several sound meaning-preserving maps on the abstract domain into a more precise one, to get a meaning-preserving lower closure operator.

DEFINITION 10.4. *Let $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ be a complete lattice and $f \in L \to L$. Define*

$$\mathfrak{I}(f) \triangleq \lambda x \bullet \bigsqcap \left\{ f(y) \;\middle|\; x \sqsubseteq y \right\}. \qquad \blacksquare$$

$\mathfrak{I}(f)$ is the greatest increasing operator on L less than or equal to f (dual of [Cousot 1978, Th. 2.4.0.2]) and so $\mathfrak{I}$ is a lower closure operator on $L \mapsto L$ such that $\mathfrak{I}(L \mapsto L) = L \xrightarrow{\nearrow} L$ is the complete lattice of increasing operators on L ordered pointwise. We observe that, if f is a meaning-preserving map, then so is $\mathfrak{I}(f)$. Because $\mathfrak{I}(f) \mathrel{\dot\sqsubseteq} f$, it is a more precise meaning-preserving map in the abstract.

LEMMA 10.5. *If γ preserves existing greatest lower bounds*[19] *and f is a meaning-preserving map (i.e. $\gamma(f(x)) = \gamma(x)$) then $\mathfrak{I}(f)$ is a meaning-preserving map (i.e. $\gamma(\mathfrak{I}(f)(x)) = \gamma(x)$).* □

PROOF.

$$\begin{array}{lll}
& \gamma(\mathfrak{I}(f)(x)) & \\
= & \gamma(\bigsqcap \left\{ f(y) \;\middle|\; x \sqsubseteq y \right\}) & \wr \text{def. } \mathfrak{I} \wr \\
= & \bigsqcap \left\{ \gamma(f(y)) \;\middle|\; x \sqsubseteq y \right\} & \wr \gamma \text{ preserves glbs} \wr \\
= & \bigsqcap \left\{ \gamma(y) \;\middle|\; x \sqsubseteq y \right\} & \wr f \text{ is a meaning-preserving map so } \gamma(f(x)) = \gamma(x) \wr \\
= & \gamma(\bigsqcap \{ y \;|\; x \sqsubseteq y \}) & \wr \gamma \text{ preserves glbs} \wr \\
= & \gamma(x) & \wr \text{def. glb} \wr \quad \square
\end{array}$$

DEFINITION 10.6. *Let $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ be a complete lattice and $f \in L \to L$. Define*

$$\mathfrak{R}(f) \triangleq \lambda x \bullet x \sqcap f(x). \qquad \blacksquare$$

By the dual of [Cousot 1978, Th. 4.2.3.0.3], $\mathfrak{R}(f)$ is the greatest reductive operator on L less than or equal to f and so $\mathfrak{R}$ is a lower closure operator on $L \mapsto L$ such that $\mathfrak{R}(L \mapsto L) = L \xrightarrow{\downarrow} L$ is the complete lattice of reductive operators on L ordered pointwise. We observe that if f is a meaning-preserving map then so is $\mathfrak{R}(f)$. Because $\mathfrak{R}(f) \mathrel{\dot\sqsubseteq} f$, it is a more precise meaning-preserving map in the abstract.

LEMMA 10.7. *If γ preserves existing greatest lower bounds and f is a meaning-preserving map (i.e. $\gamma(f(x)) = \gamma(x)$) then $\mathfrak{R}(f)$ is a meaning-preserving map (i.e. $\gamma(\mathfrak{R}(f)(x)) = \gamma(x)$).* □

PROOF.

$$\begin{array}{lll}
& \gamma(\mathfrak{R}(f)(x)) & \\
= & \gamma(x \sqcap f(x)) & \wr \text{def. } \mathfrak{R} \wr \\
= & \gamma(x) \sqcap \gamma(f(x)) & \wr \gamma \text{ preserves greatest lower bounds} \wr \\
= & \gamma(x) \sqcap \gamma(x) & \wr f \text{ is a meaning-preserving map} \wr \\
= & \gamma(x) & \wr \text{def. glb} \wr \quad \square
\end{array}$$

Given a meaning-preserving map f, it can be improved as $\mathfrak{I} \circ \mathfrak{R}(f) = \mathfrak{R} \circ \mathfrak{I}(f)$. However to get a normal form of abstract properties, we need a meaning-preserving map that is idempotent (since the normal form of a normal form is itself). The normal form can be obtained by iterating the

[19] Equivalently, γ is the upper adjoint of a Galois connection.

meaning-preserving map f or better $\mathfrak{I} \circ \mathfrak{R}(f) = \mathfrak{R} \circ \mathfrak{I}(f)$. Because theorem 10.3 shows that the limit may not be meaning-preserving, we study sufficient conditions for the limit of a meaning-preserving increasing reduction to be meaning-preserving. We also consider in corollary **10.13** the more general case of combining multiple meaning-preserving maps into a single meaning-preserving map.

LEMMA 10.8. *If* $\langle L, \leqslant, \bot, \top, \sqcap, \sqcup\rangle$ *is a complete lattice*[20] *and* $f \in L \xrightarrow{\nearrow} L$ *is increasing then* $\mathfrak{C}(f) \triangleq \boldsymbol{\lambda} x \in L \bullet \boldsymbol{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} y \bullet x \sqcap f(y)$ *is the* $\dot{\leqslant}$*-greatest lower closure on* L *less than or equal to* f.

PROOF. —— $f \in L \xrightarrow{\nearrow} L$ is increasing and $\langle L, \leqslant\rangle$ is a complete lattice so $\mathfrak{C}(f)x \triangleq \mathbf{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} y \bullet x \sqcap f(y) = \bigsqcup\{y \mid y \leqslant x \sqcap f(y)\} = \bigsqcup\{y \mid y \leqslant x \wedge y \leqslant f(y)\}$ is well-defined by Tarski fixpoint theorem [Tarski 1955].

—— Let us first prove that $\mathfrak{C}(f)$ is a lower closure on the complete lattice $\langle L, \leqslant\rangle$ that is reductive, increasing and idempotent.

— We have $\mathfrak{C}(f)x = x \sqcap f(\mathfrak{C}(f)x) \leqslant x$ proving $\mathfrak{C}(f)$ to be reductive.

— If $x \leqslant y$ then $\boldsymbol{\lambda} z \bullet x \sqcap f(z) \mathrel{\dot{\leqslant}} \boldsymbol{\lambda} z \bullet y \sqcap f(z)$ so $z \leqslant x \sqcap f(z)$ implies $z \leqslant y \sqcap f(z)$ proving $\mathfrak{C}(f)x = \mathbf{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} z \bullet x \sqcap f(z) = \bigsqcup\{z \mid z \leqslant x \sqcap f(z)\} \leqslant \bigsqcup\{z \mid z \leqslant y \sqcap f(z)\} = \mathbf{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} z \bullet y \sqcap f(z) = \mathfrak{C}(f)y$ so $\mathfrak{C}(f)$ is increasing.

— To prove that $\mathfrak{C}(f)$ is idempotent, we observe that $\mathfrak{C}(f)$ is reductive and increasing so $\mathfrak{C}(f)(\mathfrak{C}(f)(x)) \leqslant \mathfrak{C}(f)(x)$. Since $\mathfrak{C}(f)$ is reductive, $\mathfrak{C}(f)x \leqslant x$ and $\mathfrak{C}(f)x = x \sqcap f(\mathfrak{C}(f)x) \leqslant f(\mathfrak{C}(f)x)$ so $\mathfrak{C}(f)x \in \{y \mid y \leqslant x \wedge y \leqslant f(y)\}$ proving $\mathfrak{C}(f)x \leqslant \bigsqcup\{y \mid y \leqslant \mathfrak{C}(f)x \wedge y \leqslant f(y)\} = \mathfrak{C}(f)(\mathfrak{C}(f)x)$ that is $\mathfrak{C}(f)(\mathfrak{C}(f)(x)) \leqslant \mathfrak{C}(f)(x)$ by antisymmetry.

— If $f \mathrel{\dot{\leqslant}} g$ then $\boldsymbol{\lambda} y \bullet x \sqcap f(y) \mathrel{\dot{\leqslant}} \boldsymbol{\lambda} y \bullet x \sqcap g(y)$ so $\mathfrak{C}(f) \triangleq \boldsymbol{\lambda} x \in L \bullet \mathbf{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} y \bullet x \sqcap f(y) \mathrel{\dot{\leqslant}} \boldsymbol{\lambda} x \in L \bullet \mathbf{gfp}^{\leqslant}_{\top}\ \boldsymbol{\lambda} y \bullet x \sqcap g(y) \triangleq \mathfrak{C}(g)$ proving $\mathfrak{C}$ to be $\dot{\leqslant}$-increasing.

—— Let us prove that if ρ is a lower closure on $\langle L, \leqslant\rangle$ then $\mathfrak{C}(\rho) = \rho$. We have

$$
\begin{array}{lll}
& \mathfrak{C}(\rho)x & \\
= & \bigsqcup\{y \mid y \leqslant x \wedge y \leqslant \rho(y)\} & \wr \text{def. } \mathfrak{C}(\rho) \text{ and [Tarski 1955]} \wr \\
= & \bigsqcup\{y \mid y \leqslant x \wedge y = \rho(y)\} & \wr \rho \text{ reductive and antisymmetry, reflexivity for the reciprocal} \wr \\
= & \rho(x) & \wr \text{since } \rho(x) \in \{y \mid y \leqslant x \wedge y = \rho(y)\} \text{ since } \rho \text{ is reductive and idempotent, } y \leqslant x \wedge y = \rho(y) \\
 & & \text{implies } y \leqslant \rho(x) \text{ since } \rho \text{ is increasing, and def. lub } \bigsqcup \wr
\end{array}
$$

—— Finally, if ρ is a lower closure on $\langle L, \leqslant\rangle$ such that $\rho \mathrel{\dot{\leqslant}} f$ then $\rho = \mathfrak{C}(\rho) \mathrel{\dot{\leqslant}} \mathfrak{C}(f)$ since $\mathfrak{C}$ is the identity for lower closures and is increasing proving that $\mathfrak{C}(f)$ is the $\dot{\leqslant}$-greatest lower closure on L less than or equal to f. □

Here are two additional characterizations of the $\dot{\leqslant}$-greatest lower closure $\mathfrak{C}(f)$ on L less than or equal to f when $f \in L \xrightarrow{\downarrow} L$ is increasing and reductive. Given posets $\langle L, \sqsubseteq\rangle$ and $\langle P, \leqslant\rangle$, we let $L \xrightarrow{\downarrow} P$ be the set of increasing and reductive maps of L into P.

LEMMA 10.9. *If* $\langle L, \leqslant, 0, \wedge\rangle$ *is a complete lattice*[21] *and* $f \in L \xrightarrow{\downarrow} L$ *is increasing and reductive then* $\mathfrak{C}(f) = \boldsymbol{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda} g \in L \xrightarrow{\downarrow} L \bullet g \circ g$.

PROOF. —— Because $\langle L, \leqslant, 1, \vee\rangle$ is a complete lattice, $\langle L \xrightarrow{\downarrow} L, \mathrel{\dot{\leqslant}}, \boldsymbol{\lambda} x \bullet 1, \mathrel{\dot{\wedge}}\rangle$ is a complete lattice pointwise.

—— $g \circ g \in (L \xrightarrow{\downarrow} L)$ since the composition of increasing and reductive functions is increasing and reductive.

—— Let us prove that $\boldsymbol{\lambda} g \bullet g \circ g \in (L \xrightarrow{\downarrow} L) \mapsto (L \xrightarrow{\downarrow} L)$ is increasing. Indeed if $g_1 \mathrel{\dot{\leqslant}} g_2$ then by def. of a pointwise ordering $g_1 \circ g_2 \mathrel{\dot{\leqslant}} g_2 \circ g_2$ and $g_1 \circ g_1 \mathrel{\dot{\leqslant}} g_1 \circ g_2$ since g_1 is increasing, so by transitivity $g_1 \circ g_1 \mathrel{\dot{\leqslant}} g_2 \circ g_2$ proving that $\boldsymbol{\lambda} g \bullet g \circ g \in (L \xrightarrow{\downarrow} L) \xrightarrow{\nearrow} (L \xrightarrow{\downarrow} L)$

[20] The theorem also holds in a dual cpo.
[21] The theorem also holds in a dual cpo.

—— If $f \in L \xrightarrow{\downarrow} L$ is increasing and reductive then $f \circ f \mathbin{\dot{\leqslant}} f$ so f is a post-fixpoint of $\boldsymbol{\lambda}\, g \bullet g \circ g$ when considered as a function of $(L \xrightarrow{\downarrow} L) \xrightarrow{\nearrow} (L \xrightarrow{\downarrow} L)$

—— It follows that $\mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g$ exists and is pointwise less than or equal to f by Tarski fixpoint theorem on complete lattices [Tarski 1955].

—— Because $f \in L \xrightarrow{\downarrow} L$, $\boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g \in (L \xrightarrow{\downarrow} L) \xrightarrow{\nearrow} (L \xrightarrow{\downarrow} L)$, and $\langle L \xrightarrow{\downarrow} L,\ \dot{\leqslant},\ \boldsymbol{\lambda}\, x \bullet 1,\ \dot{\wedge} \rangle$ is a complete lattice, we have $(\mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g) \in L \xrightarrow{\downarrow} L$, which is increasing and reductive.

—— Moreover, $\rho \triangleq \mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g$ satisfies $\rho = \rho \circ \rho$ by the fixpoint property proving $\rho \triangleq \mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g$ to be idempotent hence an lower closure.

—— If ρ' is another lower closure on L less than of equal to f we have $\rho' \mathbin{\dot{\leqslant}} f$ and $\rho' = \rho' \circ \rho'$ so, by [Tarski 1955] fixpoint theorem, $\mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\downarrow} L \bullet g \circ g = \dot{\bigvee}\{g \in L \xrightarrow{\downarrow} L \mid g \mathbin{\dot{\leqslant}} f \wedge g = g \circ g\} \mathbin{\dot{\geqslant}} \rho'$ by def. lub $\dot{\bigvee}$. □

Lemma 10.10. *If $\langle L, \leqslant, 1, \wedge \rangle$ is a complete lattice*[22] *and $f \in L \xrightarrow{\downarrow} L$ is increasing and reductive then $\mathfrak{C}(f) = \boldsymbol{\lambda}\, x \bullet \mathbf{gfp}^{\leqslant}_{x} f$.*

Proof. —— Define $g \triangleq \mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g' \in L \xrightarrow{\downarrow} L \bullet g' \circ g'$. We just showed in lemma 10.9 that $g = \mathfrak{C}(f)$ is the $\dot{\leqslant}$-least closure that is greater than or equal to f.

—— Given any $x \in L$, x is a post-fixpoint of $f \in L \xrightarrow{\downarrow} L$ by reductivity. Since $\langle L, \leqslant, 1, \wedge \rangle$ is a complete lattice and f is increasing, $\mathbf{gfp}^{\leqslant}_{x} f$ exists, hence $\boldsymbol{\lambda}\, x \bullet \mathbf{gfp}^{\leqslant}_{x} f$ is well-defined.

—— Define $h(x) \triangleq \mathbf{gfp}^{\leqslant}_{x} f$. Let us prove that h is a lower closure.
— We have $h(x) = \mathbf{gfp}^{\leqslant}_{x} f \leqslant x$ so h is reductive.
— If $x \leqslant y$ then $z \leqslant x$ implies $z \leqslant y$ by transitivity so $\{z \mid z \leqslant x \wedge f(z) = z\} \supseteq \{z \mid z \leqslant y \wedge f(z) = z\}$ proving $h(x) = \mathbf{gfp}^{\leqslant}_{x} f = \bigvee\{z \mid z \leqslant x \wedge f(z) = z\} \leqslant \bigvee\{z \mid z \leqslant y \wedge f(z) = z\} = \mathbf{gfp}^{\leqslant}_{y} f = h(y)$ by the Tarski fixpoint theorem [Tarski 1955] and the definition lub $\bigvee$.
— $h(h(x)) = \mathbf{gfp}^{\leqslant}_{h(x)} f = h(x)$ since $f(h(x)) = h(x)$ and $h(x) \leqslant h(x)$ by reflexivity, proving h to be idempotent.

—— $\forall x \in L$, we have $h(x) = \mathbf{gfp}^{\leqslant}_{x} f \leqslant x$ so $h(x) = f(h(x)) \leqslant f(x)$ since f is increasing and by fixpoint property, so $h \mathbin{\dot{\leqslant}} f$. It follows that h is a lower closure on L that is less than or equal to f, so $h \mathbin{\dot{\leqslant}} g$ by lemma 10.9.

—— $\forall x \in L$, we have $g(x) \leqslant x$ since g is reductive. $g \triangleq \mathbf{gfp}^{\dot{\leqslant}}_{f}\ \boldsymbol{\lambda}\, g' \in L \xrightarrow{\downarrow} L \bullet g' \circ g'$ so $g \mathbin{\dot{\leqslant}} g$. In particular, $g(x) = g(g(x)) \mathbin{\dot{\leqslant}} f(g(x))$ by idempotence. It follows that $g(x) \leqslant \mathbf{gfp}^{\leqslant}_{x} f \triangleq h(x)$ by def. **gfp** . We conclude that $g \mathbin{\dot{\leqslant}} h$ pointwise.

—— By antisymmetry, we have shown that $h = g$. □

From lemmas 10.8, 10.9, and 10.10 we obtain a means for transforming a meaning-preserving map into a stronger one, which is a meaning-preserving lower closure.

Theorem 10.11. *If $\langle L, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle$ is a complete lattice, γ preserves existing glbs, $f \in L \to L$ is meaning-preserving (i.e. $\gamma \circ f = \gamma$) then the $\dot{\sqsubseteq}$-least lower closure $\mathfrak{C} \circ \mathfrak{R} \circ \mathfrak{I}(f)$ on L greater than or equal to f is also meaning-preserving i.e. ($\gamma \circ \mathfrak{C} \circ \mathfrak{R} \circ \mathfrak{I}(f) = \gamma$).* □

Proof. By lemmas 10.5 and 10.7, $\mathfrak{R} \circ \mathfrak{I}(f)$ is increasing and reductive and meaning preserving $\gamma \circ \mathfrak{R} \circ \mathfrak{I}(f) = \gamma \circ f$. By lemma 10.10 applied to $\mathfrak{R} \circ \mathfrak{I}(f)$ and [Cousot and Cousot 1979b], for all $x \in L$, $\mathfrak{C} \circ \mathfrak{R} \circ \mathfrak{I}(f)x$ is the limit of the decreasing iterates X^{δ}, $\delta \in \mathbb{O}$ of $\mathfrak{R} \circ \mathfrak{I}(f)$ from x. By definition of the iterates, we have $\gamma(X^0) = \gamma(x)$. For a successor ordinal, we have $\gamma(X^{\delta+1}) = \gamma(\mathfrak{R} \circ \mathfrak{I}(f)(X^{\delta})) = \gamma(X^{\delta}) = \gamma(x)$ since $\mathfrak{R} \circ \mathfrak{I}$ is meaning preserving and by the induction hypothesis.

[22] The lemma also holds for a dual cpo.

For a limit ordinal, $\gamma(X^\lambda) = \gamma(\bigsqcap_{\delta<\lambda} X^\delta) = \bigsqcap_{\delta<\lambda} \gamma(X^\delta) = \bigsqcap_{\delta<\lambda} \gamma(x) = \gamma(x)$ given the fact that γ is glb-preserving, the induction hypothesis, and the definition of glb. By transfinite induction, $\forall \delta \in \mathbb{O} : \gamma(X^\delta) = \gamma(x)$. For the limit X^ϵ, this implies $\mathfrak{C} \circ \mathfrak{R} \circ \mathfrak{I}(f)(x) = \mathbf{gfp}_x^{\dot\leqslant} \mathfrak{R} \circ \mathfrak{I}(f) = X^\epsilon$ proving $\gamma(\mathfrak{C} \circ \mathfrak{R} \circ \mathfrak{I}(f)(x)) = \gamma(x)$. □

10.2. Reduced Product of Abstractions Defined by Upper Closures

As another immediate consequence of lemmas 10.8, 10.9, and 10.10, we get the dual of [Ward 1942, Th. 4.2], [Cousot and Cousot 1979a, Th. 4.3] as well as a characterization of the lub in this lattice, extending [Cousot and Cousot 1979a, Th. 4.3].

Corollary 10.12. *$\mathfrak{C}$ is a lower closure on the complete lattice $\langle L \xrightarrow{\nearrow} L,\ \dot\leqslant \rangle$ ordered pointwise so that the set $\langle \mathfrak{C}(L \xrightarrow{\nearrow} L),\ \dot\leqslant \rangle$ of lower closures in L is a complete lattice.*

Proof. By the dual of [Monteiro and Ribeiro 1942], if R is a subset of a poset $\langle L, \sqsubseteq \rangle$ such that for any $x \in L$, the set $\left\{ y \in R \,\middle|\, y \sqsubseteq x \right\}$ contains a greatest element $\rho(x)$, then ρ is a lower closure operator and $R = \rho(L)$. So, by lemma 10.8, the subset of lower closures of $\langle L \xrightarrow{\nearrow} L,\ \dot\leqslant \rangle$ is $\mathfrak{C}(L \xrightarrow{\nearrow} L)$. Moreover by the dual of [Ward 1942, Th. 4.1], [Monteiro and Ribeiro 1942, Th. 8.2], it is a complete lattice. □

Corollary 10.13. *Let $\langle L, \leqslant, 0, 1, \vee, \wedge \rangle$ be a complete lattice. The glb of a set F of lower closures in the complete lattice of lower closures on L is*

$$\begin{aligned}
\mathfrak{C}(\dot\bigwedge F) &\triangleq \boldsymbol{\lambda}\, x \in L \bullet \boldsymbol{gfp}_1^{\leqslant}\ \boldsymbol{\lambda}\, y \bullet x \vee \bigwedge_{f \in F} f(y) && \text{(a)}\\
&= \boldsymbol{gfp}^{\dot\leqslant}_{\dot\bigwedge F}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\searrow} L \bullet g \circ g && \text{(b)}\\
&= \boldsymbol{\lambda}\, x \bullet \boldsymbol{gfp}_x^{\leqslant}\ \dot\bigwedge F && \text{(c)}\quad \square
\end{aligned}$$

Proof. Let F be a set of lower closures on L. $\dot\bigwedge F \in L \xrightarrow{\searrow} L$ is reductive and increasing (but may be not idempotent) so, by lemma 10.8, $\mathfrak{C}(\dot\bigwedge F) \triangleq \boldsymbol{\lambda}\, x \in L \bullet \mathbf{gfp}_\top^{\leqslant}\ \boldsymbol{\lambda}\, y \bullet x \sqcap (\dot\bigwedge F)(y)$ is the $\dot\leqslant$-greatest lower closure on L less than or equal to $\dot\bigwedge F$, hence to each $f \in F$ by def. of $\dot\bigwedge$. It follows that $\boldsymbol{\lambda}\, F \bullet \mathfrak{C}(\dot\bigwedge F)$ is the greatest lower bound in the poset of lower closures ordered pointwise, which is therefore a complete lattice [Ward 1942, Th. 4.2], [Cousot and Cousot 1979a, Th. 4.3]. By lemma 10.9, so is $\mathbf{gfp}^{\dot\leqslant}_{\dot\bigwedge F}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\searrow} L \bullet g \circ g$ proving that $\mathfrak{C}(\dot\bigwedge F) = \mathbf{gfp}^{\dot\leqslant}_{\dot\bigwedge F}\ \boldsymbol{\lambda}\, g \in L \xrightarrow{\searrow} L \bullet g \circ g$ by unicity of the greatest lower bound. By lemma 10.8, this is equal to $\boldsymbol{\lambda}\, x \in L \bullet \mathbf{gfp}_1^{\leqslant}\ \boldsymbol{\lambda}\, y \bullet x \wedge \bigwedge_{f \in F} f(y)$ and, by lemma 10.10, equal to $\boldsymbol{\lambda}\, x \bullet \mathbf{gfp}_x^{\leqslant}\ \bigwedge F$. □

In case F is finite in corollary **10.13**, we can replace $\dot\bigwedge F$ by $\bigcirc_{\rho \in F} \rho$ 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 $[1, n]$.

Corollary 10.14. *Let $\langle L, \leqslant, 0, 1, \vee, \wedge \rangle$ be a complete lattice and F be a finite set of lower closures on L. Then $\mathfrak{C}(\dot\bigwedge F) = \mathfrak{C}(\bigcirc F)$.*

Proof. If ρ_1 and ρ_2 are lower closures on L then, by the dual of [Cousot 1978, proposition 4.2.4.0.1], $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) = \mathfrak{C}(\rho_1 \circ \rho_2) = \mathfrak{C}(\rho_2 \circ \rho_1)$. Indeed, by corollary **10.12**, $\mathfrak{C}$ is reductive so $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) \mathbin{\dot\leqslant} \rho_1 \mathbin{\dot\wedge} \rho_2 \mathbin{\dot\leqslant} \rho_1$ and $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) \mathbin{\dot\leqslant} \rho_2$ so that, by idempotence of $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2)$ in lemma 10.8 and composition of increasing functions, $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) = \mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) \circ \mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) \mathbin{\dot\leqslant} \rho_1 \circ \rho_2$. Since ρ_2 is reductive and ρ_1 is increasing, we have $\rho_1 \circ \rho_2 \mathbin{\dot\leqslant} \rho_1$. Since ρ_1 is reductive, we have $\rho_1 \circ \rho_2 \mathbin{\dot\leqslant} \rho_2$ proving $\rho_1 \circ \rho_2 \mathbin{\dot\leqslant} \rho_1 \mathbin{\dot\wedge} \rho_2$ by def. glb. By corollary **10.12**, $\mathfrak{C}$ is increasing and idempotent so $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) = \mathfrak{C}(\mathfrak{C}(\rho_1 \circ \rho_2)) \mathbin{\dot\leqslant} \mathfrak{C}(\rho_1 \circ \rho_2) \mathbin{\dot\leqslant} \mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2)$ proving $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) = \mathfrak{C}(\rho_1 \circ \rho_2)$ by antisymmetry and so $\mathfrak{C}(\rho_1 \mathbin{\dot\wedge} \rho_2) = \mathfrak{C}(\rho_2 \circ \rho_1)$ since the glb $\dot\wedge$ is commutative.

It follows, by recurrence on the cardinality n of $F = \{\rho_1, \ldots, \rho_n\}$, which is finite, that $\mathfrak{C}(\bigcirc F) = \mathfrak{C}(\bigcirc_{\rho \in F} \rho) = \mathfrak{C}(\bigcirc_{i=1}^{n} \rho_i) \triangleq \mathfrak{C}(\rho_{\pi_1} \circ \ldots \circ \rho_{\pi_n}) = \mathfrak{C}(\dot{\bigwedge}\{\rho_{\pi_1}, \ldots, \rho_{\pi_n}\}) = \mathfrak{C}(\dot{\bigwedge}\{\rho_1, \ldots, \rho_n\}) = \mathfrak{C}(\dot{\bigwedge} F)$ is uniquely and well defined for all permutations π of $[1, n]$. □

Observe that an abstraction $\langle L, \leqslant, 0, 1, \vee, \wedge\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$ is equivalent to the abstract domain $\langle \gamma \circ \alpha(L), \leqslant\rangle$ since $A \mathrel{\vec{\in}} \gamma \circ \alpha(L)$ where $\gamma \circ \alpha$ is an upper closure so that abstract domains can be defined by upper closure [Cousot and Cousot 1979c]. In this case, the dual of corollary **10.13** provides the lub of such upper closures, that is the reduced product of abstractions understood as upper closures.

10.3. Iterated Pairwise Reduction of the Cartesian Product

The present definition of the most precise meaning-preserving reduction $\vec{\rho}$ in theorem 9.13 simultaneously involves all abstract domains of the product. Implementations of the most precise reduction $\vec{\rho}$ (if it exists) can hardly be modular since in general adding a new abstract domain to increase precision implies that the global reduction operator $\vec{\rho}$, hence the static analyzer/verifier, must be completely redesigned. On the contrary, the pairwise iterated product reduction below reduces abstract domains two by two, and so is more modular, in that the introduction of a new abstract domain only requires defining the reduction with the other, already existing, abstract domains. This provides a general algorithm for constructing/approximating reduced products by iterated pairwise reductions, which can be implemented once for all in the static analyzer/verifier.

Definition 10.15 (Iterated pairwise reduction). *Let $\langle A_i, \sqsubseteq_i\rangle$ be abstract domains with increasing concretization $\gamma_i \in A_i \xrightarrow{\wedge} L$ into the concrete domain $\langle L, \leqslant, \wedge\rangle$. Define $\vec{\gamma}(\vec{P}) \triangleq \bigwedge_{i \in \Delta} \gamma_i(\vec{P}_i)$ (as in definition 9.1).*

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 meaning-preserving reductions (so that $\forall \langle x, y\rangle \in A_i \times A_j : \rho_{ij}(\langle x, y\rangle) \sqsubseteq_{ij} \langle x, y\rangle$ and $(\gamma_i \times \gamma_j) \circ \rho_{ij} = (\gamma_i \times \gamma_j)$ [23]).

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

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

where $\vec{P}[i \leftarrow x]_i = x$ and $\vec{P}[i \leftarrow x]_j = \vec{P}_j$ when $i \neq j$.

Following definition 10.1 and theorem 10.3, define the iterated pairwise reductions $\vec{\rho}^n$, $\vec{\rho}^\lambda$, $\vec{\rho}^* \in \langle \vec{A}, \vec{\sqsubseteq}\rangle \mapsto \langle \vec{A}, \vec{\sqsubseteq}\rangle$, $n \geqslant 0$ of the Cartesian product for

$$\vec{\rho} \triangleq \bigcirc_{\substack{i,j \in \Delta, \\ i \neq j}} \vec{\rho}_{ij} \tag{17}$$

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 $[1, n]$. ■

Observe that $\vec{\rho}$ is the composition of meaning-preserving reductions. Thus, it is a meaning-preserving reduction, so theorems 10.2, 10.3, and 10.11 apply and produce over-approximations of the reduced product.

Theorem 10.16. *Assume in definition 10.15 that the limit $\vec{\rho}^*$ of the iterated reductions is well defined (in the sense of definition 10.1). Then the reductions are such that $\forall \vec{P} \in \vec{A} : \forall n \in \mathbb{N}_+ : \vec{\rho}^\star(\vec{P}) \mathrel{\vec{\sqsubseteq}} \vec{\rho}^n(\vec{P}) \mathrel{\vec{\sqsubseteq}} \vec{\rho}_{ij}(\vec{P}) \mathrel{\vec{\sqsubseteq}} \vec{P}$, $i, j \in \Delta$, $i \neq j$ and meaning-preserving since $\vec{\rho}^\lambda(\vec{P})$, $\vec{\rho}_{ij}(\vec{P})$, $\vec{P} \in [\vec{P}]/_{\vec{\equiv}}$.*

If, moreover, γ preserves greatest lower bounds then $\vec{\rho}^\star(\vec{P}) \in [\vec{P}]/_{\vec{\equiv}}$. □

[23] We define $(f \times g)(\langle x, y\rangle) \triangleq \langle f(x), g(y)\rangle$.

PROOF. $\vec{\rho}$ is the composition of finitely many meaning-preserving reductions so it is itself a meaning-preserving reduction. The theorem follows immediately from theorem 10.2 for finite iterations and theorem 10.3 and lemma 10.5 for transfinite iterations when is $\vec{\gamma}$ preserves existing greatest lower bounds. □

The following theorem proves that the iterated reduction may not be as precise as the reduced product, a fact underestimated in the literature. It is nevertheless easier to implement.

THEOREM 10.17. *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 $L = \wp(\{a,b,c\})$, $A_1 = \{\emptyset, \{a\}, \top\}$, $A_2 = \{\emptyset, \{a,b\}, \top\}$, $A_3 = \{\emptyset, \{a,c\}, \top\}$, and $\vec{P} = \langle \top,\ \{a,b\},\ \{a,c\}\rangle$ 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 $\vec{\rho}^{\star}(\langle \top,\ \{a,b\},\ \{a,c\}\rangle)$ is not a minimal element of $[\langle \top,\ \{a,b\},\ \{a,c\}\rangle]/_{\vec{\equiv}}$. □

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

THEOREM 10.18. *If, in definition 10.15, the $\langle A_i, \sqsubseteq_i, \sqcup_i\rangle$, $i \in \Delta$ are complete lattices, the $\vec{\rho}_{ij}$, $i, j \in \Delta$, $i \neq j$, are lower closure operators, $\vec{\gamma}$ is glb-preserving, and $\forall \vec{P}, \vec{Q} : \left(\vec{\gamma}\left(\vec{P}\right) \subseteq \vec{\gamma}\left(\vec{Q}\right)\right) \Rightarrow \left(\exists n \geqslant 0 : \left(\dot{\prod}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)^{n}(\vec{P}) \vec{\sqsubseteq} \vec{Q}\right)$ then $\forall \vec{P} : \vec{\rho}^{\star}(\vec{P})$ is the minimum of the class $[\vec{P}]/_{\vec{\equiv}}$.* □

PROOF. By theorem 10.16, we have $\forall \vec{P} : \vec{\rho}^{\star}(\vec{P}) \in [\vec{P}]/_{\vec{\equiv}}$. Let $\vec{Q} \in [\vec{P}]/_{\vec{\equiv}}$ be any other element in the same class so that $\vec{\gamma}(\vec{\rho}^{\star}(\vec{P})) = \vec{\gamma}(\vec{P}) = \vec{\gamma}(\vec{Q})$. We have

$\vec{\gamma}(\vec{P}) \subseteq \vec{\gamma}(\vec{Q})$ ⟅reflexivity⟆

$\Rightarrow \quad \exists n \geqslant 0 : \left(\dot{\prod}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)^{n}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$ ⟅by hypothesis⟆

$\Rightarrow \quad \left(\dot{\prod}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)^{\star}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$

⟅by definition 10.1 of the iterates of the reduction, which are well-defined in a complete lattice and theorem 10.3⟆

$\Rightarrow \quad \mathbf{gfp}^{\leqslant}_{\vec{P}}\left(\dot{\prod}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right) \vec{\sqsubseteq} \vec{Q}$

⟅by definition 10.1 of the iterates and the constructive version [Cousot and Cousot 1979b] of Tarski's theorem⟆

$\Rightarrow \quad \mathfrak{C}\left(\dot{\prod}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)(\vec{P}) \vec{\sqsubseteq} \vec{Q}$ ⟅by corollary **10.13**-(c)⟆

$\Rightarrow \quad \mathfrak{C}\left(\dot{\bigcirc}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)(\vec{P}) \vec{\sqsubseteq} \vec{Q}$ ⟅by corollary **10.13**⟆

$\Rightarrow \quad \mathbf{gfp}^{\leqslant}_{\vec{P}}\left(\dot{\bigcirc}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right) \vec{\sqsubseteq} \vec{Q}$ ⟅by corollary **10.13**-(c)⟆

$\Rightarrow \quad \left(\dot{\bigcirc}_{\substack{i,j\in\Delta,\\ i\neq j}} \vec{\rho}_{ij}\right)^{\star}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$

⟅by definition 10.1 of the iterates of the reduction, which are well-defined in a complete lattice and theorem 10.3 ⟆

$\Rightarrow \quad \vec{\rho}^{\star}(\vec{P}) \vec{\sqsubseteq} \vec{Q}$ ⟅by def. (17) of $\vec{\rho}$, proving $\vec{\rho}^{\star}(\vec{P})$ to be the minimum of the class $[\vec{P}]_{/\vec{\equiv}}$.⟆ □

10.4. (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.

Lemma 10.19. *Let us consider a reduced product $\langle (\bigtimes_{i\in\Delta} A_i)_{/\equiv}, \vec{\sqsubseteq}\rangle$ 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 $\vec{\gamma}\left(\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right)_{/\vec{\equiv}}\right) = \mathsf{f}[\![\mathtt{x} := t]\!](\vec{\gamma}(\vec{P}))$ and similarly for other transformers.* □

Proof.

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

Unfortunately, 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

Lemma 10.20. $\vec{\gamma}\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!](\vec{P}_i)\right) = \vec{\gamma}\left(\bigtimes_{i\in\Delta} \bar{\mathsf{f}}_i[\![\mathtt{x} := t]\!]\right)(\vec{P})$. □

Proof.

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

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

LEMMA 10.21. $f_{\mathcal{I}}[\![\mathtt{x} := e]\!]P \subseteq Q \Leftrightarrow P \subseteq \mathsf{b}_{\mathcal{I}}[\![\mathtt{x} := e]\!]Q$. □

PROOF.

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

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 over approximation 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)

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

which defines a reduction $\dot{\vec{\rho}}$ of transformers by (17) lifting the reduction $\vec{\rho}$ to the product of higher-order abstract properties.

Example 10.22. 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 $\langle \mathtt{a} = \mathtt{b},\, \mathtt{b} \geqslant 0 \wedge \mathtt{a} \geqslant 0\rangle$ whose 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 $\langle \top,\, \mathtt{a} \geqslant 0 \wedge \mathtt{b} \geqslant 0\rangle$, which is more precise than $\langle \top,\, \mathtt{b} \geqslant 0\rangle$. ■

The backward assignments $\mathsf{b}[\![\mathtt{x} := t]\!]$ and tests $\mathsf{p}[\![\varphi]\!]$ can be similarly reduced, thus generalizing [Granger 1992]).

Example 10.23. 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 [Cousot 1999, Sect. 13.9]). ■

10.5. Widening

The widening/narrowing [Cousot and Cousot 1977] 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 10.24. The closure operation in the octagon abstract domain can be considered 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 [Miné 2006b, Fig. 25–26]). ■

10.6. 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 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 section **8**, 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 engender additional reductions.

Definition 10.25 (Observational reduced product). *For all $i \in \Delta$, let $\langle {}^iA_I^{\Sigma_O}, {}^i\sqsubseteq \rangle$, $\langle {}^iA_I^{\Sigma_{O'}}, {}^i\sqsubseteq' \rangle$ be abstract domains, Ω' be the new observables, and ${}^i\mathsf{extend}_{\Omega'} \in {}^iA_I^{\Sigma_O} \to {}^iA_I^{\Sigma_{O'}}$ be the sound extensions satisfying the conditions of definition 8.11.*

The observational Cartesian product *is*

$$ {}^{\Omega'}\bigtimes_{i\in\Delta} {}^iA_I^{\Sigma_O} \triangleq \bigtimes_{i\in\Delta} {}^i\mathsf{extend}_{\Omega'}\left({}^iA_I^{\Sigma_O}\right) $$

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

11. THE NELSON-OPPEN COMBINATION PROCEDURE AS AN OBSERVATIONAL REDUCED PRODUCT

The Nelson-Oppen procedure decides satisfiability in a combination of logical theories by exchanging equalities and disequalities. Here, we show that it to computes a reduced product after the state is enhanced with some new "observations" corresponding to alien terms.

11.1. Formula Purification

11.1.1. First Phase of 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 [Nelson and Oppen 1979] 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 [Tinelli and Harandi 1996, Sect. 2] of the Nelson-Oppen combination procedure repeatedly replaces (all occurrences of) an alien subterm $t \in \mathbb{T}(\Sigma_i)\backslash\mathbb{x}$ of a subformula $\psi[t] \notin \mathbb{C}(\Sigma_i)$ of φ with the fresh variable $x \in \mathbb{x}$. Note that the subformula $\psi[t]$ includes equality or inequality predicates $\psi[t] = (t = t')$ or $(t' = t)$. The Nelson-Oppen procedure then introduces the equation $x = t$ (*i.e.* $\varphi[\psi[t]]$ is replaced by $\varphi[\psi[x]] \wedge x = t$ and recursively applies the replacement to $\varphi[\psi[x]]$ and $x = t$). Upon termination, the quantifier-free conjunctive formula φ is transformed into a formula φ' of the form

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

$\vec{x} \triangleq \bigcup_{i=1}^n \vec{x}_i$ is the set of auxiliary variables $x_i \in \vec{x}_i$ introduced by the purification, each $t_{x_i} \in \mathbb{T}(\Sigma_i)$ is an alien subterm of φ renamed to $x_i \in \mathbb{x}$, 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.

Example 11.1 (*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 $\varphi_2 = (y = g(\mathbf{x}) \wedge z = g(y))$. ∎

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 why the Nelson-Oppen purification requires theory signatures to be disjoint 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 in section **12**.

11.1.2. 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 $\mathbb{x}_\mathcal{O} = \mathbb{x}_\mathrm{P} \cup \vec{x}$. Let Σ_P and $\Sigma_\mathcal{O}$ be the corresponding signatures of Σ. Given an interpretation $I \in \mathcal{I}$, with values $I_\mathcal{V}$, the observable naming

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

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

THEOREM 11.2. $\gamma_\mathcal{I}^\mathrm{P}(\varphi') = \boldsymbol{\gamma}_\mathcal{I}^{\Omega^\varphi,\mathrm{P}}\left({}^{\Omega^\varphi}\bigtimes_{i=1}^n \varphi_i'\right)$. □

PROOF.

$$\begin{array}{lll} & \boldsymbol{\gamma}_\mathcal{I}^{\Omega^\varphi,\mathrm{P}}\left({}^{\Omega^\varphi}\bigtimes_{i=1}^n \varphi_i'\right) & \\ = & \left\{ \langle I, \eta \rangle \in \mathfrak{R}_\mathcal{I}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega_I(x)\eta \rangle \in \gamma_\mathcal{I}^{\Sigma_\mathcal{O}}\left({}^{\Omega^\varphi}\bigtimes_{i=1}^n \varphi_i'\right) \right\} & \wr \text{def. } \boldsymbol{\gamma}_\mathcal{I}^{\Omega^\varphi,\mathrm{P}} \triangleq \boldsymbol{\gamma}_\mathcal{I}^{\Omega^\varphi} \circ \gamma_\mathcal{I}^{\Sigma_\mathcal{O}} \wr \\ = & \left\{ \langle I, \eta \rangle \in \mathfrak{R}_\mathcal{I}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega_x^\varphi(\eta) \rangle \in \bigcap_{i=1}^n {}^i\gamma_\mathcal{I}^{\Sigma_\mathcal{O}}(\varphi_i') \right\} & \\ & & \wr \text{def. } \gamma_\mathcal{I}^{\Sigma_\mathcal{O}} \text{ for the observational Cartesian product} \wr \\ = & \left\{ \langle I, \eta \rangle \in \mathfrak{R}_\mathcal{I}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I, \boldsymbol{\lambda}\, x \bullet \Omega_x^\varphi(\eta) \rangle \in \gamma_\mathcal{I}^{\Sigma_\mathcal{O}}\left(\bigwedge_{i=1}^n \varphi_i'\right) \right\} & \\ & & \wr \text{def. } \gamma_\mathcal{I}^{\Sigma_\mathcal{O}}(\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}_\mathcal{I}^{\Sigma_\mathrm{P}} \;\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_\mathcal{I}^{\Sigma_\mathcal{O}}\left(\bigwedge_{i=1}^n \varphi_i'\right) \right\} & \\ & & \wr \text{def. } \Omega_x^\varphi, \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}_\mathcal{I}^{\Sigma_\mathrm{P}} \;\middle|\; \langle I, \eta \rangle \in \gamma_\mathcal{I}^{\Sigma_\mathcal{O}}\left(\exists \vec{x} : \bigwedge_{i=1}^n \varphi_i' \wedge \bigwedge_{x\in\vec{x}} x = t_x\right) \right\} & \wr \text{def. } \gamma_\mathcal{I}^{\Sigma_\mathcal{O}} \text{ and } \models \wr \\ = & \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} : \bigwedge_{i=1}^n \varphi_i' \wedge \bigwedge_{x\in\vec{x}} x = t_x\right) \right\} & \\ & & \wr \text{Since } \Sigma_\mathrm{P} \subseteq \Sigma_\mathcal{O} \text{ and } \exists \vec{x} : \textstyle\bigwedge_{i=1}^n \varphi_i' \wedge \bigwedge_{x\in\vec{x}} x = t_x \text{ has no free auxiliary variable in } \Sigma_\mathcal{O} \setminus \Sigma_\mathrm{P} \wr \end{array}$$

$$= \left\{ \langle I,\ \eta\rangle \in \mathfrak{R}^{\Sigma_{\mathrm{P}}}_{\mathcal{I}} \;\middle|\; \langle I,\ \eta\rangle \in \gamma^{\mathrm{P}}_{\mathcal{I}}\left(\exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n}\left(\varphi'_i \wedge \bigwedge_{x_i \in \vec{x}_i} x_i = t_{x_i}\right)\right) \right\} \qquad \wr \text{def. } \vec{x} \triangleq \textstyle\bigcup_{i=1}^{n} \vec{x}_i \int$$

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

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

$$= \gamma^{\mathrm{P}}_{\mathcal{I}}(\varphi') \qquad \wr \text{def. } \varphi' = \exists\, \vec{x}_1, \ldots, \vec{x}_n : \bigwedge_{i=1}^{n} \varphi_i \,. \int \quad \square$$

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

11.2. Formula Reduction

11.2.1. Second Phase of the Nelson-Oppen Theory Combination Procedure. After purification, the Nelson-Oppen combination procedure [Nelson and Oppen 1979] includes a "reduction" phase that propagates all variable equalities $x = y$ and inequalities $x \neq y$ deducible from one component φ_i in its theory $\mathcal{T}_i$ 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 $\mathcal{T}_i$ determines 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 [Tinelli and Harandi 1996], and even more efficient for Shostak theories [Shostak 1984; McIlraith and Amir 2001]. The reduction is iterated until no new disjunction of (in)equalities is found.

11.2.2. 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, or $\mathfrak{E}(S) \triangleq \left\{ r \in \wp(S \times S) \;\middle|\; \mathbb{1}_s \subseteq r \wedge r = r^{-1} \wedge r = r \circ r \right\}$. Define the pairwise reduction

$$\rho_{ij}(\varphi_i, \varphi_j) \triangleq \langle \varphi_i \wedge E_{ij} \wedge E_{ji},\ \varphi_j \wedge E_{ji} \wedge E_{ij} \rangle \quad \text{where}$$

$$\mathrm{eq}(E) \triangleq \bigvee_{\equiv \in E}\left(\bigwedge_{x \equiv y} x = y \wedge \bigwedge_{x \not\equiv y} x \neq y\right)$$

$$E_{ij} \triangleq \bigwedge \left\{ \mathrm{eq}(E) \;\middle|\; E \subseteq \mathfrak{E}(\vec{\mathtt{x}}_{\varphi_i} \cap \vec{\mathtt{x}}_{\varphi_j}) \wedge \varphi_i \Rightarrow \mathrm{eq}(E) \right\} .$$

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 11.3. Let $\varphi \triangleq (x = \mathtt{a} \vee x = \mathtt{b}) \wedge \mathtt{f}(x) \neq \mathtt{f}(\mathtt{a}) \wedge \mathtt{f}(x) \neq \mathtt{f}(\mathtt{b})$, where $\mathtt{a}, \mathtt{b}$ and $\mathtt{f}$ are in different theories. The purification yields $\varphi \triangleq \varphi_1 \wedge \varphi_2$ where $\varphi_1 \triangleq (x = \mathtt{a} \vee x = \mathtt{b}) \wedge y = \mathtt{a} \wedge z = \mathtt{b}$ and $\varphi_2 \triangleq \mathtt{f}(x) \neq \mathtt{f}(y) \wedge \mathtt{f}(x) \neq \mathtt{f}(z)$. 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 11.4. A classical example showing that the Nelson-Oppen reduction may not be as precise as the reduced product is given by [Tinelli and Harandi 1996, 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 $\varphi_2 \triangleq \mathtt{g}(x) \neq \mathtt{g}(z) \wedge$

$\mathsf{g}(y) \neq \mathsf{g}(z)$ 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 the reduced product, which is defined at the interpretation level, would propagate it. Therefore, the pairwise reduction ought to be refined to include cardinality information, as proposed by [Tinelli and Zarba 2005]. ∎

11.2.3. 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 [Gulwani and Tiwari 2006], 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.

11.3. Formula Satisfiability

After purification and reduction, the Nelson-Oppen combination procedure [Nelson and Oppen 1979] 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 post-fixpoint and whether the specification is satisfied, see section **7.5**.

12. REDUCED PRODUCT OF LOGICAL AND ALGEBRAIC ABSTRACT DOMAINS

12.1. Combining Logical and Algebraic Abstract Domains

Static analyzers such as Astrée [Bertrane et al. 2010; Cousot et al. 2005] and Clousot [Ferrara et al. 2008] are based on an iterated pairwise reduction of a product of abstract domains over-approximating their reduced product [Cousot et al. 2008]. Since logical abstract domains under the Nelson-Oppen combination procedure are indeed an iterated pairwise reduction of a product of abstract domains over-approximating their reduced product, as shown in section **11.2**, 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 domains, 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 [Cousot et al. 2008].

Notice that the Nelson-Oppen procedure and its followers aim at so-called "soundness" and refutation completeness (for the reduction to $\mathtt{ff}$). In the theorem prover community, "soundness" here means that if the procedure answers no, then the formula is not satisfiable. In program analysis there is a slightly different notion, where soundness means that whatever the answer, it is correct, and that would mean that if the procedure here answers yes, then the formula is satisfiable. This notion of soundness, when the only answers are yes it is satisfiable or no it is not, is equivalent to the old "soundness" plus completeness. This is obtained by restricting the applicability of the procedure e.g. to stably-infinite theories [Tinelli and Harandi 1996] or other similar hypotheses on interpretations [Tinelli and Zarba 2005] to ensure that models of the various theories all have the same cardinalities, and additionally by requiring that the theories are disjoint to avoid having to reduce on properties other than [dis]equality, such as inequalities. In absence of such applicability restrictions, one can retain unsatisfiability if one component formula is unsatisfiable and abandon satisfiability

if all component formulae are satisfiable in favor of "unknown", which yields reductions that are sound (as seen in section **7.5**), although potentially not optimal.

So the classical restrictions on the Nelson-Oppen procedure unnecessarily restrict its applicability to static analysis. Lifting them yields reductions that may not be optimal but preserves the soundness of the analyses, which are imprecise anyway because the underlying problem is undecidable. Hence, abandoning refutation completeness hypotheses, broadens the applicability of SMT solvers to static analysis. Many SMT solvers already contain lots of sound, but incomplete, heuristics and therefore, in practice, no longer insist on refutation completeness.

Example 12.1. As a simple example, consider the combination of the logical domain of Presburger arithmetic (where the multiplication is inexpressible) and the domain of sign analysis (which is complete for multiplication). The abstraction of a first-order formula to a formula in Presburger arithmetic eliminates all terms of the signature not in the subsignature:

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

The abstract transformers for Presburger arithmetic 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 arithmetic.

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

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

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 arithmetic 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` even though 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 arithmetic 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. ∎

12.2. 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 12.2. Astrée [Bertrane et al. 2010; Cousot et al. 2005] found runtime errors in programs that 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 [Bradley et al. 2006]. 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 precisely machine semantics can be extremely complex. In practice, one can consider such cases as errors and stop the analysis when they are encountered. It is also possible to keep the analysis running after reporting such cases, but the meaning of the analysis would no longer be a sound approximation of all program behaviors, just an approximation of the execution traces that do not fall in that case. However, this unsoundness problem disappears when proving the absence of runtime error which eliminates from consideration unpredictable behavior after the potential error. ∎

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 those discovered by the algebraic abstract domain.

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

$$
\begin{array}{rcll}
I_{\mathcal{V}} & \triangleq & (I^1_{\mathcal{V}} \cap I^2_{\mathcal{V}}) \cup \{\mathfrak{U}\}\,^{24} & \\
I_\gamma(\mathtt{c}) & \triangleq & I^1_\gamma(\mathtt{c}) & \text{when} \quad I^1_\gamma(\mathtt{c}) = I^2_\gamma(\mathtt{c}) \\
 & \triangleq & \mathfrak{U} & \text{otherwise} \\
I_\gamma(\mathtt{f}) & \triangleq & I^1_\gamma(\mathtt{f}) & \text{when } \forall v_1 \in I_{\mathcal{V}}, \ldots, \forall v_n \in I_{\mathcal{V}} : I^1_\gamma(\mathtt{f})(v_1, \ldots, v_n) = I^2_\gamma(\mathtt{f})(v_1, \ldots, v_n) \\
 & \triangleq & \mathfrak{U} & \text{otherwise} \\
I_\gamma(\mathtt{p}) & \triangleq & I^1_\gamma(\mathtt{p}) & \text{when } \forall v_1 \in I_{\mathcal{V}}, \ldots, \forall v_n \in I_{\mathcal{V}} : I^1_\gamma(\mathtt{p})(v_1, \ldots, v_n) = I^2_\gamma(\mathtt{p})(v_1, \ldots, v_n) \\
 & \triangleq & \mathfrak{U} & \text{otherwise}
\end{array}
$$

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

The main consequence is that, in absence of any error $\mathfrak{U}$, 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.

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

12.3. 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.

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 conditionals. 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, $e[x]$ is obtained from $e[t]$ by replacing all occurrences of the alien subterm t by the fresh variable x in e, then 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 has 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, $a[x]$ is obtained from $a[t]$ by replacing all occurrences of the alien subterm t by the fresh variable x, then recursively applying the replacement to $x = t$ and $\mathtt{x} := a[x]$ until no alien subterm remains.

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 before the test, then recursively purifying the assignments $x := a$.

Example 12.3. Assume that $f \in \mathbb{f}_1$ and $g \in \mathbb{f}_2$. The purification is

$$
\begin{array}{ll}
 & \mathtt{if}\ (g(\mathtt{w}) = f(g(g(\mathtt{w})))\ \mathtt{then} \ldots \\
\rightarrow & x := (g(\mathtt{w}) = f(g(g(\mathtt{w})))); \mathtt{if}\ x\ \mathtt{then} \ldots \\
\rightarrow & y := g(\mathtt{w}); x := (y = f(g(y))); \mathtt{if}\ x\ \mathtt{then} \ldots \qquad \text{⟅}g(\mathtt{w}) \text{ has type 2 and } f(g(g(\mathtt{w}))) \text{ has type 1}\text{⟆} \\
\rightarrow & y := g(\mathtt{w}); z := g(y); x := (y = f(z)); \mathtt{if}\ x\ \mathtt{then} \ldots \\
 & \qquad\qquad \text{⟅}(y = f(g(y))) \text{ has type 1 and } g(y) \text{ has type 2 .}\text{⟆}
\end{array}
$$

■

After purification all program terms are 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 $\mathbb{x}_O = \mathbb{x}_\mathtt{P} \cup \vec{x}$ be the program variables $\mathbb{x}_\mathtt{P}$ 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

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

This program transformation provides a simple implementation of the observational product of definition 10.25. Moreover, the logical abstract domains no longer need to perform purification.

THEOREM 12.4. *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 that do not have $\mathtt{e}$ in their theory $\mathcal{T}_k$, the transformer 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 $\exists x : \Psi$ and therefore does not introduce new variables to observe. For the other domains, $\mathtt{e}$ is pure so no purification is needed. The handling of backward assignment is similar and tests in a purified program only contain boolean formulæ and do not require purification 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. Here, we use existentials and remain precise by asserting the equality between those variables. The transformer is then $\exists x'_i : P_i[\mathtt{x} \leftarrow x'_i] \wedge x''_i = \mathtt{e}[\mathtt{x} \leftarrow x'_i] \wedge \exists x'_j : P_j[\mathtt{x} \leftarrow x'_j] \wedge x''_j = \mathtt{e}[\mathtt{x} \leftarrow x'_j] \wedge \ldots \wedge \mathtt{x} = x''_i = x''_j \ldots$

12.4. 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 new component);
— the downgrade consists in a pairwise reduction of a 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 complements the use of variable packs in relational abstract domains [Bertrane et al. 2010; Cousot et al. 2005], 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 a downgrade).

12.5. 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 (section **12.3**) 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 (section **12.2**),
— only introduce each domain wherever necessary (section **12.4**),
— and finally 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.

13. RELATED WORK

SMT solvers have been used in abstract interpretation, e.g. to implement specific logical abstract domains such as uninterpreted functions [Gulwani and Necula 2007] or to automatically design transformers in presence of a best abstraction [Reps et al. 2004].

Contrary to the logical abstract interpretation framework developed by [Gulwani and Tiwari 2006; Tiwari and Gulwani 2007; Gulwani et al. 2008] 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 incurring a prohibitive performance penalty (e.g. by encoding modular arithmetic in integer arithmetic or encoding floats either bitwise or with reals and rounding). So, our approach allows the description of the abstraction mechanism, comparisons of logical abstract domains, and proofs of soundness on a formal basis.

Specific combinations of theories have been proposed for static analysis such as linear arithmetic and uninterpreted functions [Gulwani and Tiwari 2006], universally quantified formulæ over theories such as linear arithmetic and uninterpreted functions [Gulwani et al. 2008] or the combination of a shape analysis with a numerical analysis [Gulwani et al. 2009] [25]. We have proposed a framework for combining algebraic and logical abstract domains on which static analyzers can be built and, incrementally and with minimal efforts, extended to 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.

14. 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 and, more generally, safety property verification/inference. The key points were to consider an observational semantics with multiple interpretations and the understanding of the Nelson-Oppen theory combination procedure [Nelson and Oppen 1979] and its followers, as well as consequence finding in structured theories [McIlraith and Amir 2001], as an iterated reduction of the product of theories. It follows 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 facilitate reuse of developments in and cooperation between both communities.

Acknowledgments

We thank Dejan Jovanović for help and Andreas Podelski for comments. Supported in part by the NSF Expeditions in Computing grant CMACS.

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

REFERENCES

Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., and Rival, X. 2010. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*. AIAA, Atlanta, Georgia, AIAA 2010–3385.

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

Bradley, A., Manna, Z., and Sipma, H. 2006. What's decidable about arrays? In *Proc. 7th Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI 2006)*, E. Emerson and K. Namjoshi, Eds. LNCS 3855, Springer, Heidelberg, Charleston, 427–442.

Chang, C. and Keisler, H. 1990. Model theory. In *Studies in logic and the foundation of mathematics* Third Ed., J. Barwise, H. J. Keisler, P. Suppes, and A. S. Troelstra, Eds. Vol. 73. Elsevier Science, New York.

Chen, L., Miné, A., Wang, J., and Cousot, P. 2011. Linear absolute value relation analysis. In *20th European Symposium on Programming (ESOP 2011), Saarbrücken, Germany*, G. Barthe, Ed. Lecture Notes in Computer Science Series, vol. 6602. Springer-Verlag, Heidelberg, 156–175.

Cook, S. 1978. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7,* 1, 70–90.

Cooper, D. 1972. Theorem proving in arithmetic without multiplication. *Machine Intelligence 91,* 7, 91–99.

Cousot, P. 1978. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Ph.D. thesis, Thèse d'État ès sciences mathématiques, Université Joseph Fourier, Grenoble, France.

Cousot, P. 1990. Methods and logics for proving programs. In *Formal Models and Semantics*, J. van Leeuwen, Ed. Handbook of Theoretical Computer Science Series, vol. B. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, Chapter 15, 843–993.

Cousot, P. 1999. The calculational design of a generic abstract interpreter, invited chapter. In *Calculational System Design*, M. Broy and R. Steinbrüggen, Eds. Vol. 173. NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, Amsterdam, 421–505.

Cousot, P. 2002. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science 277,* 1—2, 47–103.

Cousot, P. and Cousot, R. 1976. Static determination of dynamic properties of programs. In *Proc. 2nd Int. Symp. on Programming*. Dunod, Paris, Paris, 106–130.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*. ACM Press, Los Angeles, 238–252.

Cousot, P. and Cousot, R. 1979a. A constructive characterization of the lattices of all retractions, pre-closure, quasi-closure and closure operators on a complete lattice. *Portugaliæ Mathematica 38,* 2, 185–198.

Cousot, P. and Cousot, R. 1979b. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics 82,* 1, 43–57.

Cousot, P. and Cousot, R. 1979c. Systematic design of program analysis frameworks. In *6th POPL*. ACM Press, San Antonio, 269–282.

Cousot, P. and Cousot, R. 1992a. Abstract interpretation frameworks. *Journal of Logic and Computation 2,* 4, 511–547.

Cousot, P. and Cousot, R. 1992b. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *Proc. 4th International Symposium on Programming Language Implementation and Logic Programming, PLILP '92*, M. Bruynooghe and M. Wirsing, Eds. Leuven, 26–28 August 1992, LNCS 631. Springer, Heidelberg, 269–295.

Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. 2005. The Astrée analyser. In *Proc. 14th European Symp. on Programming Languages and Systems, ESOP '2005, Edinburg*, M. Sagiv, Ed. LNCS 3444. Springer, Heidelberg, 21–30.

Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. 2008. Combination of abstractions in the Astrée static analyzer. In *Eleventh Annual Asian Computing Science Conference, ASIAN 06*, M. Okada and I. Satoh, Eds. LNCS 4435, Springer, Heidelberg, Tokyo, 6–8 December 2006, 272–300.

Cousot, P., Cousot, R., and Mauborgne, L. 2010. A scalable segmented decision tree abstract domain. In *Pnueli Festschrift*, Z. Manna and D. Peled, Eds. Lecture Notes in Computer Science Series, vol. 6200. Springer-Verlag, Heidelberg, 72–95.

Craig, W. 1957. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic 22,* 3, 269–285.

Cytron, R., Ferrante, J., Rosen, B., Wegman, M., and Zadeck, F. 1991. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems 13,* 4, 451–490.

de Moura, L., Ruess, H., and Sorea, M. 2003. Bounded model checking and induction: From refutation to verification. In *Proc. 15th Computer-Aided Verification conf. (CAV'03)*, A. Voronkov, Ed. LNCS Series, vol. 2725. Springer, Heidelberg, Boulder, CO, USA, 14–26.

Detlefs, D., Nelson, G., and Saxe, J. 2005. Simplify: a theorem prover for program checking. *Journal of the ACM (JACM) 52,* 3, 365–473.

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

Elder, M., Gopan, D., and Reps, T. 2010. View-augmented abstractions. *ENTCS 267,* 1, 43–57.

Ferrante, J. and Geiser, J. 1977. An efficient decision procedure for the theory of rational order. *Theoretical Computer Science 4,* 2, 227–233.

Ferrante, J. and Rackoff, C. 1975. A decision procedure for the first order theory of real addition with order. *SIAM Journal of Computation 4,* 1, 69–76.

Ferrara, P., Logozzo, F., and Fähndrich, M. 2008. Safer unsafe code in .NET. In *Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008*, G. E. Harris, Ed. ACM Press, Nashville, TN, USA, 329–346.

Floyd, R. 1967. Assigning meaning to programs. In *Pro. Symp. in Applied Mathematics*, J. Schwartz, Ed. Vol. 19. American Mathematical Society, Providence, RI, 19–32.

Ganzinger, H. 1996. Saturation-based theorem proving (abstract). In *Proc. 23rd Int. Col., ICALP '96*, F. Meyer auf der Heide and B. Monien, Eds. LNCS 1099, Springer, Heidelberg, Paderborn, Germany, 1–3.

Ge, Y., Barrett, C., and Tinelli, C. 2007. Solving quantified verification conditions using satisfiability modulo theories. In *Conf. on Automated Deduction, CADE 21*. LNAI Series, vol. 4603. Springer, Bremen, Germany, 167–182.

Ge, Y. and de Moura, L. 2009. Complete instantiation of quantified formulas in satisfiability modulo theories. In *Computer Aided Verification, CAV'2009*. LNCS Series, vol. 5643. Springer, Grenoble, France, 306–320.

Goubault, E., Martel, M., and Putot, S. 2002. Asserting the precision of floating-point computations: A simple abstract interpreter. In *Proceedings of the 11th European Symposium on Programming, ESOP 2002*, D. Le Métayer, Ed. Lecture Notes in Computer Science Series, vol. 2305. Springer, Grenoble, France, 209–212.

Granger, P. 1989. Static analysis of arithmetical congruences. *Int. J. Comput. Math. 30,* 3 & 4, 165–190.

Granger, P. 1992. Improving the results of static analyses of programs by local decreasing iterations. In *Proceedings of the Twelfth Foundations of Software Technology and Theoretical Computer Science Conference*, R. Shyamasundar, Ed. Lecture Notes in Computer Science Series, vol. 652. Springer, Heidelberg, New Delhi, 68–79.

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

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

Gulwani, S. and Necula, G. C. 2007. Path-sensitive analysis for linear arithmetic and uninterpreted functions. In *Proceedings of the 11th International Symposium on Static Analysis, SAS '04*, R. Giacobazzi, Ed. Lecture Notes in Computer Science Series, vol. 3148. Springer, Verona, Italy, 328–343.

Gulwani, S. and Tiwari, A. 2006. Combining abstract interpreters. In *PLDI 2006*, M. Schwartzbach and T. Ball, Eds. ACM Press, Ottawa, Ontario, Canada, 376–386.

Hoare, C. 1974. Monitors: an operating system structuring concept. *Comm. ACM 17,* 10, 549–557.

Martel, M. 2009. Program transformation for numerical precision. In *Proc. 2009 ACM SIGPLAN Symp. on Partial Evaluation and Semantics-based Program Manipulation, PEPM 2009*, G. Puebla and G. Vidal, Eds. ACM, Savannah, GA, 101–110.

Mauborgne, L. 1998. Abstract interpretation using typed decision graphs. *Science of Computer Programming 31,* 1, 91–112.

McIlraith, S. and Amir, E. 2001. Theorem proving with structured theories. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, August 4–10, 2001*, B. Nebel, Ed. Morgan Kaufmann, Seattle, Washington, USA, 624–634.

McMillan, K. 2002. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification, CAV'2002*, E. Brinksma and K. Larsen, Eds. LNCS Series, vol. 2404. Springer, Heidelberg, Copenhagen, Denmark, 250–264.

McMillan, K. 2003. Craig interpolation and reachability analysis. In *Proc. 10th Int. Symp. on Static Analysis, SAS '03*, R. Cousot, Ed. LNCS 2694. Springer, Heidelberg, San Diego, CA, USA, 336.

Mendelson, E. 1997. *Introduction to mathematical logic* 4th Ed. Chapman & Hall, London.

Miné, A. 2006a. Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics. In *Proc. ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems, LCTES '2006*. ACM Press, Ottawa, Canada, 54–63.

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

Monk, J. D. 1969. *Introduction to Set Theory*. McGraw-Hill, New York.

Monteiro, A. and Ribeiro, H. 1942. L'opération de fermeture et ses invariants dans les systèmes partiellement ordonnés. *Portugal. Math. 3,* 3, 171–184.

NELSON, G. AND OPPEN, D. 1979. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems 1,* 2, 245–257.

POIZAT, B. 2000. *A Course in Model Theory: An Introduction to Contemporary Mathematical Logic*. Springer, Heidelberg.

PRATT, V. 1977. Two easy theories whose combination is hard. Tech. rep., MIT. september 1,. `boole.stanford.edu/pub/sefnp.pdf`.

RANZATO, F. 1999. Closures on CPOs form complete lattices. *Information and Computation 152*, 236–249.

REPS, T., SAGIV, S., AND YORSH, G. 2004. Symbolic implementation of the best transformer. In *Proc. 5th Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI 2004)*, B. Steffen and G. Levi, Eds. LNCS 2937, Springer, Heidelberg, Venice, Italy, 252–266.

SHOSTAK, R. 1984. Deciding combinations of theories. *Journal of the ACM 31,* 1, 1–12.

TARSKI, A. 1955. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5*, 285–310.

TINELLI, C. AND HARANDI, M. 1996. A new correctness proof of the Nelson–Oppen combination procedure. In *Frontiers of Combining Systems: Proc. 1st Int. Workshop*, F. Baader and K. U. Schulz, Eds. Applied Logic. Kluwer Academic Publishers, Munich, Germany, 103–120.

TINELLI, P. AND ZARBA, C. 2005. Combining non-stably infinite theories. *Journal of Automated Reasoning 34,* 3, 209–238.

TIWARI, A. AND GULWANI, S. 2007. Logical interpretation: Static program analysis using theorem proving. In *Automated Deduction – CADE-21*, F. Pfenning, Ed. LNCS 4603. Springer, Heidelberg, Bremen, Germany, 147–166.

WARD, M. 1942. The closure operators of a lattice. *Annals of Mathematics 43,* 2, 191–196.