The Reduced Product of Abstract Domains and the Combination of Decision Procedures

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

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

Abstract. 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 and computation on-the-fly of property transformers. It is very expressive. We propose a combination of the two approaches to reach the sweet spot best adapted to a specific application domain in the precision/cost spectrum. The proposed combination uses 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). We also introduce a semantics based on multiple interpretations to deal with the soundness of that combinations on a formal basis.

1 Introduction

Recent progress in SMT solvers and theorem provers as used in program verification [2] has been recently exploited for static analysis by abstract interpretation [3,4] using logical abstract domains [21,10]. This approach hardly scales up and is based on a mathematical program semantics quite different from the implementation (such as integers instead of modular arithmetics). Static analyzers such as ASTRÉE [1] which are based on algebraic abstractions of the machine semantics do not have such efficiency and soundness limitations. However their expressivity is limited by that of their abstract domains. It is therefore interesting to combine algebraic and logical abstract interpretations to get the best of both worlds i.e. scalability, expressivity, natural interface with the end-user using logical formulæ, and soundness with respect to the machine semantics. The proposed combination is based on the understanding of the Nelson-Oppen procedure [15] as an iterated observation reduced product.

After some syntax, we recall *multi-interpreted* semantics [5], a necessary mean to describe the soundness and relative precision of the *logical abstract domains* defined in sect. **2.8**. Next section, we introduce *observational semantics*, which is a new construction generalizing static analysis practices and necessary to describe the first step

M. Hofmann (Ed.): FOSSACS 2011, LNCS 6604, pp. 456—472, 2011.

of the Nelson-Oppen procedure in the abstract interpretation framework. Sect. **4** recalls the notion of reduced product and introduces the iterated reduced product, with new incompleteness results on that approach. Then sect. **5** is focused on the Nelson-Oppen procedure and the links with abstract interpretation. Finally, sect. **6** develops new methods to combine classical abstract interpretation and theorem provers.

2 Syntax and semantics of programs

In this section, we recall the notions introduced in [5] necessary to deal with classical abstract domains and logical abstract domains in a common semantic framework.

2.1 Syntax

We define a *signature* as a tuple $\Sigma = \langle \mathbb{x}, \mathbb{f}, \mathbb{p} \rangle$ such that the sets $\mathtt{x} \in \mathbb{x}$ of variables, $\mathtt{f} \in \mathbb{f} = \bigcup_{n \geqslant 0} \mathbb{f}^n$ of function symbols ($\mathtt{c} \in \mathbb{f}^0$ are constants), and $\mathtt{p} \in \mathbb{p} \triangleq \bigcup_{n \geqslant 0} \mathbb{p}^n$ of predicate symbols are mutually disjoints. The *terms* $t \in \mathbb{T}(\Sigma) ::= \mathtt{x} \mid \mathtt{c} \mid \mathtt{f}(t_1, \ldots, t_n)$. Conjunctive clauses $\varphi, \psi, \ldots \in \mathbb{C}(\Sigma) ::= a \mid \varphi \wedge \varphi$ are quantifier-free formulæ in simple conjunctive normal form. First-order logic formulæ $\Psi, \Phi, \ldots \in \mathbb{F}(\Sigma) ::= a \mid \neg\Psi \mid \Psi \wedge \Psi \mid \exists\, \mathtt{x} : \Psi$ may be quantified. Finally programs of the programming language on a given signature Σ are built out of basic expressions $e \in \mathbb{E}(\Sigma) \triangleq \mathbb{T}(\Sigma) \cup \mathbb{A}(\Sigma)$ and imperative commands $C \in \mathbb{L}(\Sigma)$ including assignments and tests $C ::= \mathtt{x} := e \mid \varphi$. Tests appear in conditionals and loops whose syntax, as well as that of programs, is irrelevant.

2.2 Interpretations

An *interpretation* I for a signature Σ is a pair $\langle I_{\mathcal{V}},\ I_\gamma \rangle$ such that $I_{\mathcal{V}}$ is a non-empty set of values and I_γ interprets constants, functions (into $I_{\mathcal{V}}$) and predicates (into Boolean values $\mathcal{B} \triangleq \{\mathit{false}, \mathit{true}\}$). Let $\mathfrak{I}(\Sigma)$ be the class of all such interpretations. In a given interpretation $I \in \mathfrak{I}(\Sigma)$, an environment $\eta \in \mathcal{R}_I^\Sigma \triangleq \mathbb{x} \to I_{\mathcal{V}}$ is a function from variables to values. An interpretation I and an environment $\eta \in \mathcal{R}_I^\Sigma$ *satisfy* a formula Ψ, written $I \models_\eta \Psi$, in the following way:

$$\begin{array}{rlrl} I \models_\eta a & \triangleq [\![a]\!]_I \eta & I \models_\eta \Psi \wedge \Psi' & \triangleq (I \models_\eta \Psi) \wedge (I \models_\eta \Psi') \\ I \models_\eta \neg\Psi & \triangleq \neg(I \models_\eta \Psi) & I \models_\eta \exists\, \mathtt{x} : \Psi & \triangleq \exists\, v \in I_{\mathcal{V}} :\ I \models_{\eta[\mathtt{x} \leftarrow v]} \Psi \end{array}$$ [1]

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

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

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

$$[\![\mathtt{x}]\!]_I \eta \triangleq \eta(\mathtt{x}) \qquad [\![\mathtt{c}]\!]_I \eta \triangleq I_\gamma(\mathtt{c}) \qquad [\![\mathtt{f}(t_1, \ldots, t_n)]\!]_I \eta \triangleq I_\gamma(\mathtt{f})([\![t_1]\!]_I \eta, \ldots, [\![t_n]\!]_I \eta)\,.$$

In addition, in first-order logics with equality the interpretation of equality is always $I \models_\eta t_1 = t_2 \triangleq [\![t_1]\!]_I \eta =_I [\![t_2]\!]_I \eta$ where $=_I$ is the unique reflexive, symmetric, and transitive relation on $I_{\mathcal{V}}$ encoded by its characteristic function.

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

2.3 Multi-interpreted Program Semantics

A *multi-interpreted semantics* [5] assigns meanings to a program P in the context of a set of interpretations $\mathcal{I} \subseteq \mathfrak{I}(\Sigma)$ for the program signature Σ. For example, integers can have a mathematical interpretation or a modular interpretation on machines. Then a program property in $\mathfrak{P}^\Sigma_\mathcal{I}$ provides for each interpretation in $\mathcal{I}$, a set of environments for variables $\mathbb{x}$ satisfying that property in that interpretation.

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

The multi-interpreted concrete semantics $C^\Sigma_\mathcal{I}[\![\mathtt{P}]\!] \in \mathfrak{P}^\Sigma_\mathcal{I}$ of a program P in the context of multi-interpretations $\mathcal{I}$ is assumed to be defined in least fixpoint form $C^\Sigma_\mathcal{I}[\![\mathtt{P}]\!] \triangleq \mathbf{lfp}^\subseteq F^\Sigma_\mathcal{I}[\![\mathtt{P}]\!]$ where the concrete transformer $F^\Sigma_\mathcal{I}[\![\mathtt{P}]\!] \in \mathfrak{P}^\Sigma_\mathcal{I} \stackrel{\nearrow}{\to} \mathfrak{P}^\Sigma_\mathcal{I}$ is assumed to be increasing[2]. Since $\langle \mathfrak{P}^\Sigma_\mathcal{I},\, \subseteq,\, \emptyset,\, \mathfrak{R}^\Sigma_\mathcal{I},\, \cup,\, \cap\rangle$ is a complete lattice, $\mathbf{lfp}^\subseteq F^\Sigma_\mathcal{I}[\![\mathtt{P}]\!]$ does exist by Tarski's fixpoint theorem. The transformer $F^\Sigma_\mathcal{I}[\![\mathtt{P}]\!]$ is defined by structural induction on the program P in terms of the complete lattice operations and the following local transformers for the

$$\begin{array}{lr} \text{assignment postcondition} & \mathsf{f}_\mathcal{I}[\![\mathtt{x} := e]\!]P \triangleq \left\{ \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \,\middle|\, I \in \mathcal{I} \wedge \langle I,\, \eta\rangle \in P \right\} \\ \text{assignment precondition} & \mathsf{b}_\mathcal{I}[\![\mathtt{x} := e]\!]P \triangleq \left\{ \langle I,\, \eta\rangle \,\middle|\, I \in \mathcal{I} \wedge \langle I,\, \eta[\mathtt{x} \leftarrow [\![e]\!]_I\eta]\rangle \in P \right\} \\ \text{and tests} & \mathsf{p}_\mathcal{I}[\![\varphi]\!]P \triangleq \left\{ \langle I,\, \eta\rangle \in P \,\middle|\, I \in \mathcal{I} \wedge [\![\varphi]\!]_I\eta = \mathit{true} \right\} . \end{array}$$

To recover the usual concrete semantics, let $\mathfrak{I} \in \mathfrak{I}(\Sigma)$ be the *program standard interpretation* e.g. as defined explicitly by a standard or implicitly by a compiler, linker, loader, operating system and network of machines. Then the standard concrete semantics is $\mathcal{I} = \{\mathfrak{I}\}$. The reason why we consider multi-interpretations is that it is the natural setting for the logical abstract domains which are valid up to a theory (Sect. **2.7**), which can have many different interpretations.

2.4 Algebraic Abstract Domains

We let $\langle A^\Sigma_\mathcal{I},\, \sqsubseteq,\, \top,\, \sqcup,\, \ldots,\, \bar{\mathsf{f}}, \bar{\mathsf{p}},\, \ldots\rangle$ be an *abstract domain* abstracting multi-interpreted properties in $\mathfrak{P}^\Sigma_\mathcal{I}$ for signature Σ and multi-interpretations $\mathcal{I}$ with partial ordering $\sqsubseteq$. Pre-orders are assumed to be quotiented by the preorder equivalence so $A^\Sigma_\mathcal{I}$ is a poset but may be not a complete lattice nor a cpo. The meaning of the abstract properties is defined by an increasing *concretization* function $\gamma^\Sigma_\mathcal{I} \in A^\Sigma_\mathcal{I} \stackrel{\nearrow}{\to} \mathfrak{P}^\Sigma_\mathcal{I}$. In case of existence of a best abstraction, we use a *Galois connection* $\langle \mathfrak{P}^\Sigma_\mathcal{I},\, \subseteq\rangle \underset{\alpha^\Sigma_\mathcal{I}}{\overset{\gamma^\Sigma_\mathcal{I}}{\leftrightarrows}} \langle A^\Sigma_\mathcal{I},\, \sqsubseteq\rangle$ [4].

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

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

[2] f is increasing (or monotone) if $x \preceq y$ implies $f(x) \sqsubseteq f(y)$, written $f \in \langle P,\, \preceq\rangle \stackrel{\nearrow}{\to} \langle Q,\, \sqsubseteq\rangle$.

The concrete least fixpoint semantics $C^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!]$ of a program P in Sect. **2.3** may have no correspondent in the abstract e.g. because the abstract domain $\langle A^{\Sigma}_{\mathcal{I}}, \sqsubseteq\rangle$ is not a cpo so that the abstract transformer has no least fixpoint, even maybe no fixpoint. In that case, we can define the abstract semantics $\overline{C}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] \in \wp(A^{\Sigma}_{\mathcal{I}})$ as the set of abstract inductive invariants for an abstract transformer $\overline{F}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] \in A^{\Sigma}_{\mathcal{I}} \to A^{\Sigma}_{\mathcal{I}}$ of P.

$$\overline{C}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!] \triangleq \left\{ \overline{P} \in A^{\Sigma}_{\mathcal{I}} \;\middle|\; \overline{F}^{\Sigma}_{\mathcal{I}}[\![\mathtt{P}]\!](\overline{P}) \sqsubseteq \overline{P} \right\} \quad \text{postfixpoint semantics.}$$

In practice, only one abstract postfixpoint needs to be computed (while the abstract semantics defines all possible ones). Such an abstract postfixpoint can be computed e.g. by elimination or iteratively from the infimum using widenings and narrowings [3].

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

2.5 Soundness and Completeness of Abstract Semantics

The abstract semantics $\overline{C}[\![\mathtt{P}]\!] \in A$ is *sound* with respect to a concrete semantics $C[\![\mathtt{P}]\!]$ of a program P for concretization γ whenever $\forall \overline{P} \in A : (\exists\, \overline{C} \in \overline{C}[\![\mathtt{P}]\!] :\ \overline{C} \sqsubseteq \overline{P}) \Rightarrow (C[\![\mathtt{P}]\!] \subseteq \gamma(\overline{P}))$. It is *complete* whenever $\forall \overline{P} \in A : (C[\![\mathtt{P}]\!] \subseteq \gamma(\overline{P})) \Rightarrow (\exists\, \overline{C} \in \overline{C}[\![\mathtt{P}]\!] : \overline{C} \sqsubseteq \overline{P})$. When the concrete semantics is defined in fixpoint form $C[\![\mathtt{P}]\!] \triangleq \mathbf{lfp}^{\subseteq} F[\![\mathtt{P}]\!]$ and the abstract semantics in postfixpoints, the soundness of the abstract semantics follows from the soundness conditions of the abstraction in Sect. **2.4** and the soundness of the abstract transformer $\forall \overline{P} \in A : F[\![\mathtt{P}]\!] \circ \gamma(\overline{P}) \subseteq \gamma \circ \overline{F}[\![\mathtt{P}]\!](\overline{P})$ [3,4]. If the concrete semantics is also defined in postfixpoint form, then the soundness condition becomes

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

Moreover, the composition of sound abstractions is necessarily sound.

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

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

2.6 Abstractions between Multi-interpretations

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

If we can only compute properties on the standard interpretation $\mathfrak{I}$ then we can approximate a multi-interpreted program saying that we know the possible behaviors when the interpretation is $\mathfrak{I}$ and we know nothing (so all properties are possible) for

the other interpretations of the program. On the other hand, if we analyze a program that can only have one possible interpretation with a multi-interpreted property, then we are doing an abstraction in the sense that we add more behaviors and forget the actual property that should be associated with the program by the standard semantics. So, in general, we have two sets of interpretations, one is $\mathcal{I}$, the context of interpretations for the program and the other one is $\mathcal{I}^\sharp$, the set of interpretations used in the analysis. The correspondance between the two is a Galois connection.

Lemma 1. $\langle \mathfrak{P}^\Sigma_{\mathcal{I}}, \subseteq\rangle \underset{\alpha^\Sigma_{\mathcal{I}\to\mathcal{I}^\sharp}}{\overset{\gamma^\Sigma_{\mathcal{I}^\sharp\to\mathcal{I}}}{\leftrightarrows}} \langle \mathfrak{P}^\Sigma_{\mathcal{I}^\sharp}, \subseteq\rangle$ *with* $\gamma^\Sigma_{\mathcal{I}^\sharp\to\mathcal{I}}(P^\sharp) \triangleq \left\{ \langle I,\eta\rangle \in \mathfrak{R}^\Sigma_{\mathcal{I}} \;\middle|\; I \in \mathcal{I}^\sharp \Rightarrow \langle I,\eta\rangle \in P^\sharp \right\}$ *and* $\alpha^\Sigma_{\mathcal{I}\to\mathcal{I}^\sharp}(P) \triangleq P \cap \mathfrak{R}^\Sigma_{\mathcal{I}^\sharp}$. □

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

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 fixpoint semantics do differ when $\mathcal{I}^\sharp \neq \mathcal{I}$ and $\mathcal{I} \not\subseteq \mathcal{I}^\sharp$ since $\langle \mathfrak{P}^\Sigma_{\mathcal{I}^\sharp}, \subseteq\rangle \neq \langle \mathfrak{P}^\Sigma_{\mathcal{I}}, \subseteq\rangle$. We have soundness.

Lemma 2. $f_{\mathcal{I}}[\![\mathtt{x} := e]\!] \circ \gamma^\Sigma_{\mathcal{I}^\sharp\to\mathcal{I}}(P^\sharp) = \gamma^\Sigma_{\mathcal{I}^\sharp\to\mathcal{I}} \circ f_{\mathcal{I}^\sharp}[\![\mathtt{x} := e]\!](P^\sharp)$*, and similarly for the other transformers.* □

2.7 Theories and Models

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

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

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

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

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

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

This notion of models gives a natural way of approximating sets of interpretations by a theory: a set of interpretations $\mathcal{I}$ can be approximated by any theory $\mathcal{T}$ such that

$\mathcal{I} \subseteq \mathfrak{M}(\mathcal{T})$. Notice, though, that because the lattice of sentences of a theory is not complete, there is no best abstraction in general[3].

2.8 Logical Abstract Domains

Given a theory $\mathcal{T}$ over Σ, a *logical abstract domain* is an abstract domain $\langle A^\Sigma_\mathcal{T}, \sqsubseteq, \top, \sqcup, \ldots, \bar{\mathtt{f}}, \bar{\mathtt{p}}, \ldots\rangle$ such that $A^\Sigma_\mathcal{T} \subseteq \mathbb{F}(\Sigma)$, $\sqsubseteq \triangleq \Rightarrow$, $\top \triangleq \mathtt{tt}$, $\sqcup \triangleq \vee$, etc, and the concretization is $\gamma^\Sigma_\mathcal{T}(\Psi) \triangleq \left\{ \langle I, \eta\rangle \,\middle|\, I \in \mathfrak{M}(\mathcal{T}) \text{ and } I \models_\eta \Psi \right\}$. Note that a logical abstract domain is a special case of algebraic abstract domain over a multi-interpretation.

Remark that there might be no finite formula in the language $\mathbb{F}(\Sigma)$ of the theory $\mathcal{T}$ to encode a best abstraction in which case there is no Galois connection. In any case soundness can be formalized by a concretization function as in Sect. **2.4**. Moreover, in presence of infinite ascending chains of finite first-order formulæ (e.g. $(\mathtt{x} = 0) \Rightarrow (\mathtt{x} = 0 \vee \mathtt{x} = 1) \Rightarrow \ldots \Rightarrow \bigvee_{i=1}^n \mathtt{x} = i \Rightarrow \ldots$) and descending chains of finite formulæ (e.g. $(\mathtt{x} \neq -1) \Leftarrow (\mathtt{x} \neq -1 \wedge \mathtt{x} \neq -2) \Leftarrow \ldots \Leftarrow \bigwedge_{i=1}^n \mathtt{x} \neq -i \Leftarrow \ldots$) with no finite first-order formula to express their limits, the fixpoint may not exist. Hence the fixpoint semantics in the style of Sect. **2.3** is not well-defined in the abstract. However, following Sect. **2.4**, we can define the abstract semantics as the set of abstract inductive invariants for an increasing abstract transformer of program P.

3 Observational Semantics

Besides values of program variables, the concrete semantics may also observe values of auxiliary variables or values of functions over program variables. Whereas such cases can be described in the general setting above (e.g. by inclusion of the auxiliary variables as program variables), it is more convenient to explicitly define the observables of the program semantics.

3.1 Observable Properties of Multi-interpreted Programs

The signature $\Sigma = \langle \mathbb{x}, \mathbb{f}, \mathbb{p}\rangle$ of multiple interpretations $\mathcal{I} \in \wp(\mathfrak{I}(\Sigma))$ is decomposed into a program signature $\Sigma_\mathtt{P} = \langle \mathbb{x}_\mathtt{P}, \mathbb{f}, \mathbb{p}\rangle$ over program variables $\mathtt{x} \in \mathbb{x}_\mathtt{P} \subseteq \mathbb{x}$ and an observable signature $\Sigma_\mathcal{O} = \langle \mathbb{x}_\mathcal{O}, \mathbb{f}, \mathbb{p}\rangle$ over observable identifiers $x \in \mathbb{x}_\mathcal{O} \subseteq \mathbb{x}$. So we now have

program variables	observable variables	
$\eta \in \mathcal{R}^{\Sigma_\mathtt{P}}_I \triangleq \mathbb{x}_\mathtt{P} \to I_\mathcal{V}$	$\zeta \in \mathcal{R}^{\Sigma_\mathcal{O}}_I \triangleq \mathbb{x}_\mathcal{O} \to I_\mathcal{V}$	program environments
$\mathfrak{R}^{\Sigma_\mathtt{P}}_\mathcal{I} \triangleq \left\{ \langle I, \eta\rangle \,\middle|\, I \in \mathcal{I} \wedge \eta \in \mathcal{R}^{\Sigma_\mathtt{P}}_I \right\}$	$\mathfrak{R}^{\Sigma_\mathcal{O}}_\mathcal{I} \triangleq \left\{ \langle I, \zeta\rangle \,\middle|\, I \in \mathcal{I} \wedge \zeta \in \mathcal{R}^{\Sigma_\mathcal{O}}_I \right\}$	multi-interpreted environments
$\mathfrak{P}^{\Sigma_\mathtt{P}}_\mathcal{I} \triangleq \wp(\mathfrak{R}^{\Sigma_\mathtt{P}}_\mathcal{I})$	$\mathfrak{P}^{\Sigma_\mathcal{O}}_\mathcal{I} \triangleq \wp(\mathfrak{R}^{\Sigma_\mathcal{O}}_\mathcal{I})$	multi-interpreted properties

[3] If $\mathfrak{I}$ interprets programs over the natural numbers there is no enumerable first-order theory characterizing this interpretation (by Gödel first incompleteness theorem), so the poset has no best abstraction of $\{\mathfrak{I}\}$

We name observables by identifiers (which, in particular, can be variable identifiers). Observables are functions from values of program variables to values $v \in I_{\mathcal{V}}$ (for interpretation $I \in \mathfrak{I}(\Sigma)$).

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

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

Example 1 (Memory model). In the memory model of [14], 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 $P \in \mathfrak{P}_{\mathcal{I}}^{\Sigma_{\mathrm{P}}}$, the corresponding observable property is

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

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 $Q \in \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$, the corresponding program property is

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

We have a Galois connection between the program and observable properties.

Theorem 1. $\langle \mathfrak{P}_{\mathcal{I}}^{\Sigma_{\mathrm{P}}},\ \subseteq\rangle \underset{\boldsymbol{\alpha}_{\mathcal{I}}^{\Omega}}{\overset{\boldsymbol{\gamma}_{\mathcal{I}}^{\Omega}}{\leftrightarrows}} \langle \mathfrak{P}_{\mathcal{I}}^{\Sigma_O},\ \subseteq\rangle$. □

3.2 Soundness of the Abstraction of Observable Properties

The *observational abstraction* will be of observable properties in $\mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ so with concretization $\gamma_{\mathcal{I}}^{\Sigma_O} \in A_{\mathcal{I}}^{\Sigma_O} \to \mathfrak{P}_{\mathcal{I}}^{\Sigma_O}$ where $A_{\mathcal{I}}^{\Sigma_O}$ is the abstract domain. The classical direct abstraction of program properties in $\mathfrak{P}_{\mathcal{I}}^{\Sigma_{\mathrm{P}}}$ will be the particular case where $\mathbb{X}_O = \mathbb{X}_{\mathrm{P}}$ and $\lambda\, \mathtt{x} \bullet \Omega_I(\mathtt{x})$ is the identity. The program properties corresponding to observable Ω_I are given by $\boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}} \in A_{\mathcal{I}}^{\Sigma_O} \mapsto \mathfrak{P}_{\mathcal{I}}^{\Sigma_{\mathrm{P}}}$ such that

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

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

Lemma 3. $\boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}(\bar{f}[\![\mathtt{x} := e]\!]\overline{P}) \supseteq f_I[\![\mathtt{x} := e]\!](\boldsymbol{\gamma}_{\mathcal{I}}^{\Omega,\mathrm{P}}(\overline{P}))$ *and similarly for the other transformers.*

3.3 Observational Extension

It can sometimes be useful to extend an abstract property $\overline{P}$ for observables Ω with a new observable ω named x. For example, this was useful for intervals in [6]. We will write $\mathsf{extend}_{(x,\omega)}\left(\overline{P}\right)$ for the extension of $\overline{P}$ with observable ω for observable identifier x.

Example 2. Let $A_{\mathbb{x}_O}$ be the abstract domain mapping observable identifiers $\mathtt{x} \in \mathbb{x}_O$ to an interval of values [3]. Assume that intervals of program variables are observable, that is $\mathbb{x}_P \subseteq \mathbb{x}_O$ and let $\mathtt{x} \in \mathbb{x}_P$ be a program variables for which we want to observe the square $\mathtt{x}^2$ so $\omega_I \triangleq [\![\mathtt{x}^2]\!]_I$. Let $\mathtt{x2} \notin \mathbb{x}_O$ be a fresh name for this observable. This extension of observable properties with a new observable $\mathsf{extend}_{(\mathtt{x2},[\![\mathtt{x}^2]\!])} \in A_{\mathbb{x}_O} \to A_{\mathbb{x}_O \cup \{\mathtt{x2}\}}$ can be defined as

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

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

The extension operation is assumed to be defined so that its semantics satisfies the following soundness condition

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

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

Lemma 4. *For the logical abstract domain* $A \triangleq \mathbb{F}(\Sigma)$ *with* $\gamma_{\mathcal{I}}^{\Sigma_O}(\Psi) \triangleq \left\{ \langle I, \eta \rangle \;\middle|\; I \in \mathcal{I} \wedge I \models_\eta \Psi \right\}$,

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

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

4 Iterated Reduction and Reduced Product

A reduction makes a property more precise in the abstract without changing its concrete meaning. By iterating this reduction, one can improve the precision of a static analysis without altering its soundness. A case of iterated reduction was proposed by [8] following [4].

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

Theorem 2 (Iterated reduction). *Given a sound reduction* ρ, *for all ordinals* λ, ρ^λ *is a sound reduction. If the iterates of* ρ *from* $\overline{P}$ *are well-defined then their limit* $\rho^*(\overline{P})$ *exists. We have* $\forall \beta < \lambda : \rho^*(\overline{P}) \sqsubseteq \rho^\lambda(\overline{P}) \sqsubseteq \rho^\beta(\overline{P}) \sqsubseteq \overline{P}$. *If* γ *is the upper adjoint of a Galois connection then* ρ^* *is a sound reduction. If* ρ *is increasing then* $\rho^* = \boldsymbol{\lambda}\, \overline{P} \bullet \boldsymbol{gfp}^{\sqsubseteq}_{\overline{P}} \rho$ *is the greatest fixpoint (gfp) of* ρ *less than or equal to* $\overline{P}$. □

The reduced product is defined as follows [4].

Definition 2 (Reduced product). *Let $\langle A_i, \sqsubseteq_i \rangle$, $i \in \Delta$, Δ finite, be abstract domains with increasing concretization $\gamma_i \in A_i \to \wp^{\Sigma^o}_{\mathcal{I}}$. Their* Cartesian product *is $\langle \boldsymbol{A}, \sqsubseteq \rangle$ where $\boldsymbol{A} \triangleq \bigtimes_{i\in\Delta} A_i$, $(\boldsymbol{P} \sqsubseteq \boldsymbol{Q}) \triangleq \bigwedge_{i\in\Delta}(\boldsymbol{P}_i \sqsubseteq_i \boldsymbol{Q}_i)$ and $\boldsymbol{\gamma} \in \bigtimes_{i\in\Delta} A_i \to \wp^{\Sigma^o}_{\mathcal{I}}$ is $\boldsymbol{\gamma}(\boldsymbol{P}) \triangleq \bigcap_{i\in\Delta} \gamma_i(\boldsymbol{P}_i)$. In particular the* product *$\langle A_i \times A_j, \sqsubseteq_{ij} \rangle$ is such that $\langle x, y\rangle \sqsubseteq_{ij} \langle x', y'\rangle \triangleq (x \sqsubseteq_i x') \wedge (y \sqsubseteq_j y')$ and $\gamma_{ij}(\langle x,\ y\rangle) \triangleq \gamma_i(x) \cap \gamma_j(y)$.*

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

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

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

Definition 3 (Iterated pairwise reduction). *For $i, j \in \Delta$, $i \neq j$, let $\rho_{ij} \in \langle A_i \times A_j, \sqsubseteq_{ij} \rangle \mapsto \langle A_i \times A_j, \sqsubseteq_{ij} \rangle$ be pairwise reductions (so that $\forall \langle x, y\rangle \in A_i \times A_j : \rho_{ij}(\langle x, y\rangle) \sqsubseteq_{ij} \langle x, y\rangle$, preferably lower closure operators i.e. reductive, increasing and idempotent). Define the pairwise reductions $\boldsymbol{\rho}_{ij} \in \langle \boldsymbol{A}, \sqsubseteq \rangle \mapsto \langle \boldsymbol{A}, \sqsubseteq \rangle$ of the Cartesian product as*

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

*where $\boldsymbol{P}[i \leftarrow x]_i = x$ and $\boldsymbol{P}[i \leftarrow x]_j = \boldsymbol{P}_j$ when $i \neq j$. Define the iterated pairwise reductions $\boldsymbol{\rho}^n$, $\boldsymbol{\rho}^* \in \langle \boldsymbol{A}, \sqsubseteq \rangle \mapsto \langle \boldsymbol{A}, \sqsubseteq \rangle$, $n \geqslant 0$ of the Cartesian product as in Def. **1** for*

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

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

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

Theorem 3. *Under the hypotheses of Def. **1** and assuming the limit of the iterated reductions is well defined, the reductions are such that $\forall \boldsymbol{P} \in \boldsymbol{A} : \forall \lambda : \boldsymbol{\rho}^\star(\boldsymbol{P}) \sqsubseteq \boldsymbol{\rho}^\lambda(\boldsymbol{P}) \sqsubseteq \boldsymbol{\rho}_{ij}(\boldsymbol{P}) \sqsubseteq \boldsymbol{P}$, $i, j \in \Delta$, $i \neq j$ and sound since $\boldsymbol{\rho}^\lambda(\boldsymbol{P}), \boldsymbol{\rho}_{ij}(\boldsymbol{P}), \boldsymbol{P} \in [\boldsymbol{P}]_{/\equiv}$ and if γ preserves lower bounds then $\boldsymbol{\rho}^\star(\boldsymbol{P}) \in [\boldsymbol{P}]_{/\equiv}$.* □

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

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

Theorem 5. *If the* $\langle A_i, \sqsubseteq_i, \sqcup_i \rangle$, $i \in \Delta$ *are complete lattices, the* ρ_{ij}, $i, j \in \Delta$, $i \neq j$, *are lower closure operators, and* $\forall \boldsymbol{P}, \boldsymbol{Q} : \Big(\boldsymbol{\gamma}(\boldsymbol{P}) \subseteq \boldsymbol{\gamma}(\boldsymbol{Q})\Big) \Rightarrow \Big(\exists n \geqslant 0 : \Big(\dot{\bigsqcap}_{\substack{i,j \in \Delta, \\ i \neq j}} \boldsymbol{\rho}_{ij}\Big)^n (\boldsymbol{P}) \sqsubseteq \boldsymbol{Q}\Big)$ *then* $\forall \boldsymbol{P} : \boldsymbol{\rho}^\star(\boldsymbol{P})$ *is the minimum of the class* $\boldsymbol{P}/_{\equiv}$. □

4.1 Observational Reduced Product

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

Definition 4 (Observational reduced product). *For all* $i \in \Delta$, *let* $\langle {}^iA_{\boldsymbol{I}}^{\Sigma_O}, {}^i\sqsubseteq \rangle$, $\langle {}^iA_{\boldsymbol{I}}^{\Sigma_{O'}}, {}^i\sqsubseteq' \rangle$ *be abstract domains,* Ω' *be the new observables, and* ${}^i\mathsf{extend}_{\Omega'} \in {}^iA_{\boldsymbol{I}}^{\Sigma_O} \to {}^iA_{\boldsymbol{I}}^{\Sigma_{O'}}$ *be sound extensions in the sense that* ${}^i\boldsymbol{\gamma}_{\boldsymbol{I}}^{\Omega',\mathrm{P}}\left({}^i\mathsf{extend}_{\Omega'}\left(\overline{P}\right)\right) \supseteq {}^i\boldsymbol{\gamma}_{\boldsymbol{I}}^{\Omega,\mathrm{P}}\left(\overline{P}\right)$.

The observational cartesian product *is* ${}^{\Omega'}\bigtimes_{i \in \Delta} {}^iA_{\boldsymbol{I}}^{\Sigma_O} \triangleq \bigtimes_{i \in \Delta} {}^i\mathsf{extend}_{\Omega'}\left({}^iA_{\boldsymbol{I}}^{\Sigma_O}\right)$ *and the* observational reduced product *is* $\langle \left({}^{\Omega}\bigtimes_{i \in \Delta} A_i\right)/_{\equiv}, \sqsubseteq \rangle$. □

5 The Nelson-Oppen Combination Procedure

The Nelson-Oppen procedure which decides satisfiability in a combination of logical theories by exchanging equalities and disequalities is shown to consist in computing a reduced product after the state is enhanced with some new "observations" corresponding to alien terms.

5.1 Formula Purification

Formula Purification in the Nelson-Oppen Theory Combination Procedure Given disjoint deductive theories $\mathcal{T}_i$ in $\mathbb{F}(\Sigma_i)$, $\Sigma_i \subsetneq \Sigma$ 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 [15] 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 [18, Sect. 2] of the Nelson-Oppen combination procedure consists in repeating the replacement of (all occurrences of) an alien subterm $t \in \mathbb{T}(\Sigma_i) \setminus \mathbb{X}$ of a subformula $\psi[t] \notin \mathbb{C}(\Sigma_i)$ (including equality or inequality predicates $\psi[t] = (t = t')$ or $(t' = t)$) of φ by a fresh variable $x \in \mathbb{X}$ and introducing the equation

$x = t$ (i.e. $\varphi[\psi[t]]$ is replaced by $\varphi[\psi[x]] \wedge x = t$ and the replacement is recursively applied to $\varphi[\psi[x]]$ and $x = t$). Upon termination, the quantifier-free conjunctive formula φ is transformed into a formula φ' of the form

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

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

Example 3 (Formula purification). Assume $f \in \mathbb{f}_1$ and $g \in \mathbb{f}_2$. $\varphi = (g(\mathbf{x}) = f(g(g(\mathbf{x}))))$ $\rightarrow (\exists y : y = f(g(y)) \wedge y = g(\mathbf{x})) \rightarrow (\exists y : \exists z : y = f(z) \wedge y = g(\mathbf{x}) \wedge z = g(y)) \rightarrow$ $(\exists y : \exists z : \varphi_1 \wedge \varphi_2) = \varphi'$ 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 the Nelson-Oppen purification requires disjointness of theory signatures is that otherwise they can share more than equalities and cardinality, a sufficient reason for the procedure to be incomplete. Nevertheless, the purification procedure remains sound for non-disjoint theories, which can be exploited for static analysis, as shown below.

The Nelson-Oppen Purification as an Observational Cartesian Product Let the observable identifiers be the free variables of $\varphi \in \mathbb{C}(\Sigma)$, $\mathbb{x}_\mathrm{P} = \mathbf{x}_\varphi$ plus the fresh auxiliary variables $\boldsymbol{x}$ introduced by the purification $\mathbb{x}_O = \mathbb{x}_\mathrm{P} \cup \boldsymbol{x}$. Let Σ_P and Σ_O be the corresponding signatures of Σ. Given an interpretation $I \in \mathcal{I}$, with values $I_\mathcal{V}$, the observable naming $\Omega_I^\varphi \in \mathbb{x}_O \rightarrow \mathcal{R}_I^{\Sigma_\mathrm{P}} \rightarrow I_\mathcal{V}$ is such that

$$\begin{aligned} \Omega_I^\varphi(x)\eta &\triangleq \eta(x) && \text{when} \quad x \in \mathbb{x}_\mathrm{P}, \\ &\triangleq [\![t_x]\!]\eta && \text{when} \quad x \in \boldsymbol{x} \quad . \end{aligned}$$

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

Theorem 6. $\gamma_I^\mathrm{P}(\varphi') = \boldsymbol{\gamma}_I^{\Omega^\varphi, \mathrm{P}}\left({}^{\Omega^\varphi}\bigtimes_{i=1}^{n} \varphi'_i\right)$. □

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

5.2 Formula Reduction

Formula Reduction in the Nelson-Oppen Theory Combination Procedure After purification, the Nelson-Oppen combination procedure [15] includes a reduction phase

where all variable equalities $x = y$ and inequalities $x \neq y$ deducible from one component φ_i in its theory $\mathcal{T}_i$ are propagated to all components φ_j.The decision procedure for $\mathcal{T}_i$ is used to determine all possible disjunctions of conjunctions of (in)equalities that are implied by φ_i. These are determined by exhaustively trying all possibilities in the nondeterministic version of the procedure or by an incremental construction in the deterministic version, which is more efficient for convex theories [18]. The reduction is iterated until no new disjunction of (in)equalities is found.

The Nelson-Oppen Reduction as an Iterated Fixpoint Reduction of the Product Let $\mathfrak{E}(S)$ be the set of all equivalence relations on S. Define the pairwise reduction $\rho_{ij}(\varphi_i, \varphi_j) \triangleq \langle \varphi_i \wedge E_{ij} \wedge E_{ji},\ \varphi_j \wedge E_{ji} \wedge E_{ij} \rangle$ where

$$\mathrm{eq}(E) \triangleq \bigvee_{\equiv \in E} \left(\bigwedge_{x \equiv y} x = y \wedge \bigwedge_{x \not\equiv y} x \neq y \right) \text{ and } E_{ij} \triangleq \bigwedge \left\{ \mathrm{eq}(E) \;\middle|\; E \subseteq \mathfrak{E}(\mathtt{x}_{\varphi_i} \cap \mathtt{x}_{\varphi_j}) \wedge \varphi_i \Rightarrow \mathrm{eq}(E) \right\} .$$

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

Example 4. Let $\varphi_1 \triangleq (x = \mathtt{a} \vee x = \mathtt{b}) \wedge y = \mathtt{a} \wedge z = \mathtt{b}$ and $\varphi_2 \triangleq \mathtt{f}(x) \neq \mathtt{f}(y) \wedge \mathtt{f}(x) \neq \mathtt{f}(z)$ so that $\varphi \triangleq \varphi_1 \wedge \varphi_2$ is purified. We have $E_{12} \triangleq (x = y) \vee (x = z)$ and $E_{21} \triangleq (x \neq y) \wedge (x \neq z)$ so that $\boldsymbol{\rho}^*(\varphi) = \mathtt{ff}$. □

Example 5. A classical example showing that the Nelson-Oppen reduction may not be as precise as the reduced product is given by [18, p. 11] where $\varphi_1 \triangleq \mathtt{f}(x) \neq \mathtt{f}(y)$ in the theory of Booleans admitting models of cardinality at most 2 and $\varphi_2 \triangleq \mathtt{g}(x) \neq \mathtt{g}(z) \wedge \mathtt{g}(y) \neq \mathtt{g}(z)$ in a disjoint theory admitting models of any cardinality so that $\varphi = \varphi_1 \wedge \varphi_2$ is purified. The reduction yields $\varphi \wedge x \neq y \wedge x \neq z \wedge y \wedge z$ and not $\mathtt{ff}$ since the cardinality information is not propagated whereas it would be propagated by the reduced product which is defined at the interpretation level. Therefore the pairwise reduction ought to be refined to include cardinality information, as proposed by [20]. □

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 [12], 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.

5.3 Formula Satisfiability

After purification and reduction, the Nelson-Oppen combination procedure [15] 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 $\mathbb{ff}$. The satisfiability decision can also be used as an approximation to check for a postfixpoint and that the specification is satisfied.

For briefness, we have concentrated in this paper on the Nelson-Oppen combination procedure [15] but Shostak combination procedure [17] can be handled in exactly the same way. The idea of iterated reduction also applies to theorem proving [13].

6 Reduced Product of Logical and Algebraic Abstract Domains

6.1 Combining Logical and Algebraic Abstract Domains

Static analyzers such as Astrée [1] and Clousot [7] are based on an iterated pairwise reduction of a product of abstract domains over-approximating their reduced product. Since logical abstract domains as combined by 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 Sect. **5.2**, the design of abstract interpreters based on an approximation of the reduced product can use both logical and algebraic abstract domains.

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

Notice that the Nelson-Oppen procedure and its followers aim at so-called "soundness" and refutation completeness (for the reduction to $\mathbb{ff}$). In the theorem prover community, "soundness" here means that if the procedure answers no, then the formula is not satisfiable. In program analysis we have 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 [18] or other similar hypotheses on interpretations [20] to ensure that models of the various theories all have the same cardinalities, and additionaly by requiring that the theories are disjoint to avoid having to reduce on other properties than [dis]equality. In absence of such applicability restrictions, one can retain unsatisfiability if one component formula is unsatisfiability and abandon satisfiability if all component formula are satisfiable in favor of “unknown”, which yields reductions that are sound 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 have to be imprecise anyway by undecidability. Hence, abandoning refutation completeness hypotheses, broaden the

applicability of SMT solvers to static analysis. Many SMT solvers already contain lots of sound, but incomplete, heuristics hence no longer insist on refutational completeness.

Example 6. As a simple example, consider the combination of the logical domain of Presburger arithmetics (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 of Presburger arithmetics is by abstraction to a subsignature eliminating all terms of the signature not in the subsignature:

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

$$\alpha_\Sigma(\neg\Psi) \triangleq \neg\alpha_\Sigma(\Psi) \qquad \alpha_\Sigma(\Psi \wedge \Psi') \triangleq \alpha_\Sigma(\Psi) \wedge \alpha_\Sigma(\Psi')) \qquad \alpha_\Sigma(\exists\, \mathtt{x} : \Psi) \triangleq \exists\, \mathtt{x} : \alpha_\Sigma(\Psi)\,.$$

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

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

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

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

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

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

6.2 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 pu-

rification, 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 conditional or iteration tests. A term $t \in \mathbb{T}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i)$ not reduced to a variable is said "*to have type i*" when it is of the form $\mathtt{c} \in \mathbb{f}_i^0$ or $\mathtt{f}(t_1, \ldots, t_n)$ with $\mathtt{f} \in \mathbb{f}_i^n$. As a side note, one may observe that this could very well be equivalent to using the variable and term types in a typed language.

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

An atomic formula $a \in \mathbb{A}(\mathbb{x}, \bigcup_{i=1}^n \mathbb{f}_i, \bigcup_{i=1}^n \mathbb{p}_i)$ not reduced to $\mathtt{false}$ is said *to have type i* when it is of the form $\mathtt{p}(t_1, \ldots, t_n)$ with $\mathtt{p} \in \mathbb{p}_i^n$ or $t_1 = t_2$ and t_1 has type i or $\mathtt{x} = t_2$ and t_2 has type i. Then we can purify an assignment $\mathtt{x} := a[t]$ exactly in the same way as with terms. Finaly the purification of a clause in a test consists in replacing each atomic subformula a of the clause by a fresh variable and introducing assignments $x := a$ before the test and in recursively purifying the assignments $x := a$.

Example 7. Assume that $f \in \mathbb{f}_1$ and $g \in \mathbb{f}_2$. The purification is

$$
\begin{aligned}
& \mathtt{if}\ (g(\mathtt{w}) = f(g(g(\mathtt{w})))\ \mathtt{then} \ldots \\
\rightarrow\ & x := (g(\mathtt{w}) = f(g(g(\mathtt{w})))); \mathtt{if}\ x\ \mathtt{then} \ldots \\
\rightarrow\ & y := g(\mathtt{w}); x := (y = f(g(y))); \mathtt{if}\ x\ \mathtt{then} \ldots \\
& \qquad\qquad \wr g(\mathtt{w}) \text{ has type 2 and } f(g(g(\mathtt{w}))) \text{ has type 1} \int \\
\rightarrow\ & y := g(\mathtt{w}); z := g(y); x := (y = f(z)); \mathtt{if}\ x\ \mathtt{then} \ldots \\
& \qquad\qquad \wr (y = f(g(y))) \text{ has type 1 and } g(y) \text{ has type 2 .} \int \quad \square
\end{aligned}
$$

After purification all program terms and clauses are pure in that no term 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 $t \in \mathbb{T}(\Sigma_O^i)$ 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 $\mathbb{x}_O = \mathbb{x}_\mathtt{P} \cup \boldsymbol{x}$ be the program variables $\mathbb{x}_\mathtt{P}$ plus the fresh auxiliary variables $x \in \boldsymbol{x}$ introduced by the purification with assignments $x := e_x$. Given an interpretation I, with values $I_\mathcal{V}$, the observable naming $\Omega_I \in \mathbb{x}_O \mapsto (\mathbb{x}_\mathtt{P} \mapsto I_\mathcal{V}) \mapsto I_\mathcal{V}$ is

$$
\begin{aligned}
\Omega_I(x)\eta &\triangleq \eta(x) \quad \text{when} \quad x \in \mathbb{x}_\mathtt{P} \\
&\triangleq [\![e_x]\!]\eta \quad \text{when} \quad x \in \boldsymbol{x}\ .
\end{aligned}
$$

This program transformation provides a simple implementation of the observational product of Def. **4**. Moreover, the logical abstract domains no longer need to perform purification.

Theorem 7. *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. □

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 e in their language, so that we can use existentials and remain precise by asserting the equality between thoses variables.

7 Related Work

SMT solvers have been used in abstract interpretation, e.g. to implement specific logical abstract domains such as uninterpreted functions [11] or to automatically design transformers in presence of a best abstraction [16].

Contrary to the logical abstract interpretation framework developed by [12,21,10] we do not assume that the behavior of the program is described by formulæ in the same theory as the theory of the logical abstract domain, which offers no soundness guarantee, but instead we give the semantics of the logical abstract domains with respect to a set of possible semantics which includes the possibility of a sound combination of a mathematical semantics and a machine semantics, which is hard to achieve in SMT solvers without breaking down their performances (e.g. by encoding modular arithmetics in integer arithmetics or encoding floats either bitwise or with reals and roundings). So, our approach allows the description of the abstraction mechanism, comparisons of logical abstract domains, and to provide proofs of soundness on a formal basis.

Specific combinations of theories have been proposed for static analysis such as linear arithmetic and uninterpreted functions [12], universally quantified formulæ over theories such as linear arithmetic and uninterpreted functions [10] or the combination of a shape analysis with a numerical analysis [9][4]. The framework that we propose to combine algebraic and logical abstract domains can be used to design static analyzers incrementally, with minimal efforts to include new abstractions to improve precision either globally for the whole program analysis or locally, e.g. to prove loop invariants provided by the end user.

8 Conclusion

We have proposed a new design method of static analyzers based on the reduced product or its approximation by the iterated reduction of the product to combine algebraic and logical abstract domains. This is for invariance inference but is also applicable to invariant verification. The key points were to consider an observational semantics with multiple interpretations and the understanding of the Nelson-Oppen theory combination procedure [15] and its followers, as well as consequence finding in structured theories [13], as an iterated reduction of the product of theories so that algebraic and logical abstract domains can be symmetrically combined in a product either reduced or with iterated reduction. The interest of the (reduced) product in logical abstract interpretation

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

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.

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

Acknowledgments We thank D. Jovanović and A. Podelski for help and comments.

References

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

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

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th POPL*, 238–252, 1977.

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th POPL*, 269–282, 1979.

[5] P. Cousot, R. Cousot and L. Mauborgne. Logical Abstract Domains and Interpretations. In *The Future of Engineering*, S. Nanz (Ed.), Springer, 2010.

[6] M. Elder, D. Gopan and T. Reps. View-Augmented Abstractions. *2nd NSAD*, ENTCS, 2010.

[7] P. Ferrara, F. Logozzo, and M. Fähndrich. Safer unsafe code in .NET. *OOPSLA*, 329–346, 2008.

[8] P. Granger. Improving the results of static analyses of programs by local decreasing iterations. *FST and TCS*, LNCS 652, 68–79, 1992.

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

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

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

[12] S. Gulwani and A. Tiwari. Combining abstract interpreters. *PLDI*, 376–386, 2006.

[13] S.A. McIlraith and E. Amir. Theorem proving with structured theories. *IJCAI*, 624–634, 2001.

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

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

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

[17] N. Shankar and H. Rueß. Combining Shostak theories. *Rewriting Techniques and Applications*, LNCS 2378, 1–18, 2002.

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

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

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

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