Compositionality in the puzzle of semantics

Roberto Giacobazzi
Dipartimento di Informatica
Università di Verona
Strada Le Grazie, I-37134 Verona, Italy
giaco@sci.univr.it

Isabella Mastroeni
Dipartimento di Informatica
Università di Verona
Strada Le Grazie, I-37134 Verona, Italy
mastroeni@sci.univr.it

ABSTRACT

In this paper we study the connection between the structure of relational abstract domains for program analysis and compositionality of the underlying semantics. Both can be systematically designed as solution of the same abstract domain equation involving the same domain refinement: the reduced power operation. We prove that most well-known compositional semantics of imperative programs, such as the standard denotational and weakest precondition semantics can be systematically derived as solutions of simple abstract domain equations. This provides an equational presentation of both semantics and abstract domains for program analysis in a unique formal setting. Moreover both finite and transfinite compositional semantics share the same structure, and this allows us to provide consistent models for program manipulation.

Categories and Subject Descriptors

D.3 [**Programming languages**]: Formal definitions and theory—*Semantics*; F.3 [**Logics and meanings of programs**]: Semantics of Programming Languages—*algebraic approaches to semantics, denotational semantics, operational semantics*

General Terms

Theory, Verification

Keywords

Abstract interpretation, reduced power, compositional semantics, transfinite semantics, program manipulation.

1. INTRODUCTION

Compositionality plays a key role in program manipulation and program analysis. Compositional semantics are typically used to derive compositional analyzers, where the analysis of a program can be obtained by composing the analysis of its sub-components or to derive program manipulation algorithms. The essence of compositional semantics is usually hidden in the ability of the semantics to observe input/output relations about program's behaviors. One of the best frameworks to study semantics and compare them according to their relative precision is the hierarchy developed in [8], here called Cousot's hierarchy of semantics, where semantics at different levels of abstractions are related with each other by abstract interpretation. A number of semantics including big-step, termination and non-termination, Plotkin's natural, Smyth's demonic, Hoare's angelic relational and corresponding denotational, Dijkstra's predicate transformer weakest-precondition and weakest-liberal precondition and Hoare's partial and total axiomatic semantics, have all been derived by successive abstractions starting from an (operational) maximal trace semantics of a transition system. The resulting hierarchy provides a complete account on the structure and the relative precision of most well-known semantics of programming languages.

One of the major challenge in Cousot's construction is that *semantics are abstract domains*. Therefore they can be transformed, refined, decomposed, and composed similarly to what is usually done with abstract domains in static program analysis [18]. In this paper we characterize compositional semantics as solutions of simple abstract domain equations. The idea is to consider the *reduced power* operation [9] for abstract domain refinement as the basic operation able to include input/output relations in domains. The reduced power operation has been proved to give the necessary structure of abstract domains in order to model relational properties of program behavior [10, 21, 24, 28]. A similar structure can be included in semantics as well. If $[\![P_1]\!]$ and $[\![P_2]\!]$ are the semantics of program components P_1 and P_2, and $\diamond$ is a syntactic operator for program composition, then the semantics $[\![\cdot]\!]$ is compositional if there exists an operation $\circ$ such that: $[\![P_1 \diamond P_2]\!] = [\![P_1]\!] \circ [\![P_2]\!]$. We show that most well-known compositional semantics of imperative programs, such as the standard angelic denotational and weakest-liberal precondition semantics, can be systematically derived as solutions of simple abstract domain equations. We consider sequential syntactic composition of programs and trace composition $\frown$ for composing semantics. In this case compositionality boils down to $[\![P_1; P_2]\!] = [\![P_1]\!] \frown [\![P_2]\!]$. We prove that the compositional semantics observing finite computations only, such that the angelic denotational and weakest-liberal precondition semantics, can be systematically derived as the most abstract semantics closed under reduced power and including non-

PEPM '02, Jan. 14-15, 2002 Portland, OR, USA

compositional semantics which observe respectively final and initial states of finite traces. Unfortunately this construction is not applicable to infinite semantics. This is mainly due to the fact that the composition of infinite traces looses precision as we are unable to observe limit states, when they exist, on which traces may compose. These facts put in evidence that the use of reduced power in these conditions is limited and it doesn't allow us to capture exactly the relational information in infinite computations. This fact leads to a stronger notion of compositionality, where the equation above holds also for non-terminating *slices* of code.

In order to find a semantics that satisfies this new concept of compositionality we define a non-standard semantics of traces, here called transfinite trace semantics, that is able to *observe* what happens also after an infinite loop[1]. The transfinite semantics of a program is the set of all its possibly transfinite computations, i.e. computations whose length can be any ordinal, finite or infinite. We insert this new semantics in the Cousot's hierarchy as a concretization (through a Galois insertion) of the maximal trace semantics. In this context we are able to systematically build by reduced power a transfinite version of denotational and weakest precondition semantics and prove that the standard corresponding semantics are abstractions of these latter ones. In this way we can outline a projection of the Cousot's (standard) hierarchy of semantics in a more concrete level, here called non-standard level, connected with the standard one by abstract interpretation (see Fig. 2). In this new context we are able to prove the optimality of denotational and weakest precondition non-standard semantics, i.e. they are the most abstract semantics, on the non-standard level, that observe respectively the last and the first state of computations and that are closed under reduced power refinement. Moreover we prove formally the compositionality, in the stronger sense, of these semantics. This peculiarity of the transfinite denotational and weakest precondition semantics allows us to use them for modeling program slicing [26, 32] and more in general in program manipulation. Moreover the fact that these semantics don't loose precision in presence of infinite computations makes it possible to give semantics to a program P in a way such that the semantics of a slice of P is always contained in the semantics of P. This proves the consistency of program slicing with respect to these semantics.

Related work

The idea of using abstract domain transformers to study semantics of programming languages is not new. For instance in [7] the domain operation of *tensor product* [30] is considered in order to design Hoare's axiomatic semantics by exploiting the adjoint relation between pre and post conditions in Hoare triples. Most of the works that apply abstract domain transformers in the design of semantics consider the logic programming paradigm, which basically relies on the hierarchy of semantics developed in [4, 15]. In [15] reduced product is used in order to systematically compose a concrete semantics with abstract domains for program analysis. In [17] the authors study the relations between different semantics of logic programs, such as success pattern semantics, computed answer substitution semantics, and call pattern semantics by means of complementation. This idea is further exploited in [16] where domain complementation is considered in order to study the symmetrical structure of Cousot's hierarchy of semantics, in particular in the characterization of the complementary nature of angelic, infinite, and demonic semantics of a transition system.
As for compositionality is concerned, the very first and, up to our knowledge, unique example of construction of compositional semantics by abstract domain transformation is in [19]. In this work, the authors proved that compositional semantics of logic programs in [2, 14] can be systematically designed by a generalization of Cousot's reduced cardinal power operation [9] from non-compositional semantics of computed answer substitution. This work represents a starting point for our work, which generalizes the results in [19] to arbitrary programming languages whose semantics can be specified by a transition system of states. Moreover, we prove that the standard and non-standard (transfinite) compositional semantics are optimal, i.e. they are the most abstract solutions of simple domain equations. Clearly, our results can be specialized to the case of logic and constraint logic programming, obtaining in this way an enhanced theory for logic programming compositionality with respect to [2, 14, 19].
As for program manipulation, and in particular program slicing, is concerned, in [1, 3, 26, 29] we can find several approaches to lazy semantics for program dependence graphs. These kind of semantics are used in order to model program manipulation in a way such that the semantics of a slice approximates the semantics of the full program.

2. PRELIMINARIES

In the following $g \circ f$ denotes function composition, i.e., $g \circ f \stackrel{\text{def}}{=} \lambda x.g(f(x))$. The identity function $\lambda x.\ x : X \longrightarrow X$ is denoted ι_X, $\cong$ denotes the isomorphism of ordered structures, and $S \longrightarrow T$ denotes the set of all functions from S to T ordered point-wise by $\sqsubseteq$. Let C and A be complete lattices. Then, $C \xrightarrow{\text{m}} A$, $C \xrightarrow{\text{c}} A$, $C \xrightarrow{\text{a}} A$, and $C \xrightarrow{\text{coa}} A$ denote, respectively, the set of all monotone, (Scott-)continuous, additive (i.e. commuting on arbitrary suprema), and coadditive functions from C to A. The first infinite ordinal is $\omega \stackrel{\text{def}}{=} \bigcup_{n\in\mathbb{N}} n$. The proper class of all the ordinals is $\mathbb{O}$. $\mathbb{O}_l$ is the proper class of limit ordinals, and λ is a typical limit ordinal. Note that $0 \in \mathbb{O}_l$.
Following standard definitions we consider abstract domains formulated either in terms of Galois connections or in terms of closure operators [9]. An *upper closure operator* on a poset $\langle P, \leq_P\rangle$ is an operator $\rho : P \to P$ monotone, idempotent and extensive ($\forall x \in P.\ x \leq_P \rho(x)$). $uco(P)$ denotes the set of all upper closure operators on P. Often, we will find particularly convenient to identify closure operators with their sets of fix-points. If $\langle C, \leq, \vee, \wedge, \top, \bot\rangle$ is a complete lattice then $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \lambda x.x\rangle$ is also a complete lattice where, for every $\{\rho\}$, $\{\eta\}$, $\{\rho_i\}_{i\in I} \subseteq uco(C)$ and $x \in C$: $\rho \sqsubseteq \eta$ iff $\forall y \in C.\ \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$ and $(\sqcap_{i\in I}\rho_i)(x) = \wedge_{i\in I}\rho_i(x)$. If $\alpha : C \xrightarrow{\text{m}} A$ and $\gamma : A \xrightarrow{\text{m}} C$ are monotone functions such that $\iota_C \sqsubseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \sqsubseteq \iota_A$, then (A, α, γ, C) is a *Galois connection* (GC) between A and C. The set of all GC's between two complete lattices A and C is the *tensor product* $A \otimes C$. $A \otimes C \cong A \xrightarrow{\text{a}} C \cong C \xrightarrow{\text{coa}} A$ [30]. If $\alpha \circ \gamma = \iota_A$ we have a *Galois insertion* (GI) also denoted $\langle C, \leq_C\rangle \underset{\gamma}{\overset{\alpha}{\leftrightarrows}} \langle A, \leq_A\rangle$. Note that ${}_\gamma A \cong C$ if and only if $\langle C, \leq_C\rangle \underset{\alpha}{\overset{\gamma\ \alpha}{\leftrightarrows}} \langle A, \leq_A\rangle$. If $\langle C, \leq_C\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq_A\rangle$ and

[1]Cousot, in [7], noted this possibility underlining the utility of a transfinite trace semantics for modeling program slicing.

$f : C \longrightarrow C$, the *best correct approximation* $f^\sharp : A \longrightarrow A$ of f is defined as $f^\sharp \stackrel{\text{def}}{=} \alpha \circ f \circ \gamma$. Note that $\alpha \circ f \leq_A f^\sharp \circ \alpha$.
Any GI $\langle C, \leq_C \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq_A \rangle$ uniquely determines an upper closure operator $\gamma \circ \alpha \in uco(C)$ and conversely $\rho \in uco(C)$ uniquely determines a GI $(\rho(C), \rho, id, C)$. Hence, we will identify $uco(C)$ with the so-called *lattice* $\mathfrak{L}_C$ *of abstract interpretations* of C [9]. The point-wise ordering on $uco(C)$ corresponds precisely to the standard ordering used to compare abstract domains with regard to their precision: A_1 is more precise than A_2 (i.e., A_2 is an abstraction of A_1) iff $A_1 \sqsubseteq A_2$ in $uco(C)$ iff $\langle A_1, \leq_{A_1} \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A_2, \leq_{A_2} \rangle$.

Reduced product and *power* are the best known operations to compose domains in order to exploit respectively the attribute independent and relational properties of programs [9]. The reduced product $\sqcap_{i \in I} A_i$ of a family of domains $\{A_i\}_{i \in I} \subseteq uco(C)$ is the most abstract domain in $\mathfrak{L}_C$ which is more concrete than every A_i. The reduced relative power in [19] is a generalization over arbitrary quantales of Cousot's original reduced power [9]. Let $\langle D, \leq, \odot \rangle$ be a *semi-quantale* [27], i.e. an algebraic structure where $\langle D, \leq \rangle$ is a complete lattice and $\odot : D \times D \longrightarrow D$ is an associative, monotone and left-additive binary operation. Given a pair of Galois connections between a concrete domain D and the two domains D_1 and D_2: $\langle D, \leq_D \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle D_1, \leq_{D_1} \rangle$ and $\langle D, \leq_D \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle D_2, \leq_{D_2} \rangle$, we define the (relative) reduced power of D_1 and D_2, as the set $D_1 \overset{\odot}{\longrightarrow} D_2$ of all the monotone functions from D_1 to D_2 defined as $\lambda x.\alpha_2(d \odot \gamma_1(x))$ with $d \in D$. The dual operation for a right-additive operation $\odot$ is $D_2 \overset{\odot}{\longleftarrow} D_1$ and it represents the set of all the monotone functions $\lambda x.\alpha_2(\gamma_1(x) \odot d)$. $D_1 \overset{\odot}{\longrightarrow} D_2$ is called *forward reduced power* and the set $D_2 \overset{\odot}{\longleftarrow} D_1$ is called *backward reduced power*. We have that $\langle D, \leq_D \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle D_1 \overset{\odot}{\longrightarrow} D_2, \sqsubseteq \rangle$ with the function $\alpha = \lambda d.\, \lambda x.\, \alpha_2(d \odot \gamma_1(x))$ and, by duality, $\langle D, \leq_D \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle D_2 \overset{\odot}{\longleftarrow} D_1, \sqsubseteq \rangle$ with $\alpha = \lambda d.\lambda x.\alpha_2(\gamma_1(x) \odot d)$.

3. COUSOT'S SEMANTICS HIERARCHY

We consider Cousot's hierarchy of semantics as depicted in Fig. 1 (see [7, 11]), where continuous lines and arrows denote, respectively, isomorphisms and strict abstractions between semantics. Given a transition system $\langle \Sigma, \tau \rangle$ with $\tau \subseteq \Sigma \times \Sigma$, Σ^+ and $\Sigma^\omega \stackrel{\text{def}}{=} \mathbb{N} \longrightarrow \Sigma$ denote respectively the set of finite non-empty and of (ω-)infinite sequences on Σ, the set of all sequences is $\Sigma^\infty = \Sigma^+ \cup \Sigma^\omega$. The length of a sequence σ is denoted $|\sigma| \in \omega$ and its i-th element is denoted σ_i. A non-empty finite (infinite) *trace* σ is a finite (infinite) sequence of program states where two consecutive elements are in the transition relation τ, i.e. for all $i < |\sigma|$: $\langle \sigma_i, \sigma_{i+1} \rangle \in \tau$. The *maximal trace semantics* of a transitions system [11] is the set τ^∞ of all the finite and infinite traces of a program. The *concatenation* of traces is defined as follows: Let $\sigma, \delta \in \Sigma^\infty$, $|\sigma| = n$, $|\delta| = m$ and $A, B \subseteq \Sigma^\infty$ then $\sigma^\frown \delta = \eta$ where $\sigma_{n-1} = \delta_0$ and $\forall i < n \,.\, \eta_i = \sigma_i \ \wedge \ \forall n \leq j < m \,.\, \eta_j = \delta_{j-n+1}$ while $A^\frown B = \left\{ \ \sigma^\frown \delta \ \middle| \ \sigma \in A \ \wedge \ \delta \in B \ \wedge \ \sigma_{n-1} = \delta_0 \ \right\}$. This operation is both right and left additive and $\forall X \in \wp(\Sigma^\infty) \,.\, X^\frown \varnothing = \varnothing$ and $\Sigma^\frown X = X^\frown \Sigma = X$.
Each semantics in Cousot's hierarchy is derived as abstract interpretation of the maximal trace semantics τ^∞, as summarized in Table 1. The *relational semantics* $\mathcal{R}^\infty$ associates an input/output relation with program traces by using the bottom symbol $\perp \notin \Sigma$ to denote non-termination. The *denotational semantics* $\mathcal{D}^\infty$ gives semantics by considering input-output functions. Dijkstra's predicate transformer $g\mathcal{W}p$ and Hoare's axiomatic semantics $g\mathcal{H}$ consider respectively the weakest precondition predicate transformers and the set of all the Galois connections that specify the adjoint relation between weakest precondition and strongest-postcondition in Hoare's triples $\{P\}\ C\ \{Q\}$ [7]. Each semantics in natural style may have a corresponding *angelic*, *demonic*, and *infinite* observable which is again an abstraction, as summarized in Table 1. The *angelic* trace semantics τ^+ observes only finite computations. We denote by $\mathcal{R}^+$, $\mathcal{D}^+$, $\mathcal{W}lp$, and $p\mathcal{H}$ the angelic abstractions of the corresponding semantics in natural style. The *demonic* semantics approximates non-termination by chaos, i.e. if there is a possibility of non-termination then any possible output is given, and this corresponds to allow the worst possible behavior of the program [8, 13]. We denote by $\mathcal{R}^\partial$, $\mathcal{D}^\partial$, $\mathcal{W}p^\partial$, and $g\mathcal{H}^\partial$ the demonic observables of the corresponding semantics in natural style. The *infinite* trace semantics τ^ω, with corresponding infinite relational $\mathcal{R}^\omega$, observes non-terminating traces only.

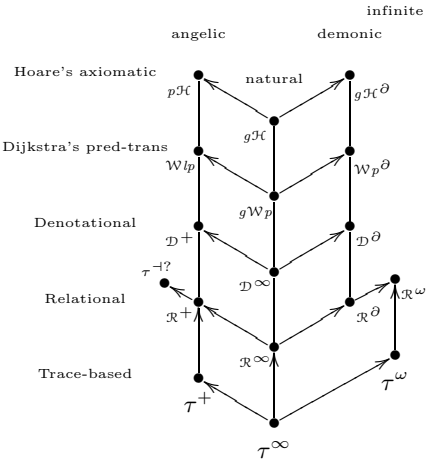

Figure 1: Semantics in Cousot's hierarchy.

4. PROGRAM MANIPULATION

In the following we consider a simple imperative language, IMP [33], with the following syntax:

$$c ::= \textbf{nil} \mid id := e \mid c; c \mid \textbf{if } e \textbf{ then } c \textbf{ else } c \mid \textbf{while } e \textbf{ do } c$$

We know that the denotational semantics is *compositional*, namely it is equal to the composition of the semantics of program's sub-components. We want to give a characterization of this property of denotational semantics as a property of the corresponding abstraction in Cousot's hierarchy. One of the major consequence of Cousot's construction of semantics as abstract interpretations is that semantics correspond to abstract domains, namely it is possible to associate with each semantics a closure operator, i.e. an abstract domain, defined on the domain of maximal finite and infinite traces, and representing what is actually observable by the semantics. Hence we can think of formulating the problem of compositionality in terms of closure operators: If X and Y are two sets of traces representing the semantics of the

Semantics	*Domain relation*	*Abstraction*
$\mathcal{R}^\infty = \alpha^{\mathcal{R}}(\tau^\infty)$	$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^{\mathcal{R}}}{\overset{\gamma^{\mathcal{R}}}{\leftrightarrows}} \langle \wp(\Sigma \times \Sigma_\perp), \subseteq\rangle$	$\alpha^{\mathcal{R}}(X) = \{\ \langle \sigma_0, \sigma_{n-1}\rangle \mid\ \sigma \in X^+\ \} \cup \{\ \langle \sigma_0, \perp\rangle \mid\ \sigma \in X^\omega\ \}$
$\mathcal{D}^\infty = \alpha^{\mathcal{D}}(\mathcal{R}^\infty)$	$\langle \wp(\Sigma \times \Sigma_\perp), \subseteq\rangle \underset{\alpha^{\mathcal{D}}}{\overset{\gamma^{\mathcal{D}}}{\leftrightarrows}} \langle \Sigma \longrightarrow \wp(\Sigma_\perp), \sqsubseteq\rangle$	$\alpha^{\mathcal{D}}(X) \stackrel{\text{def}}{=} \lambda s.\{s' \in \Sigma_\perp \mid \langle s, s'\rangle \in X\}$
$g\mathcal{W}p = \alpha^{g\mathcal{W}p}(\mathcal{D}^\infty)$	$\langle \Sigma \longrightarrow \wp(\Sigma_\perp), \sqsubseteq\rangle \underset{\alpha^{g\mathcal{W}p}}{\overset{\gamma^{g\mathcal{W}p}}{\leftrightarrows}} \langle \wp(\Sigma_\perp) \overset{\text{coa}}{\longrightarrow} \wp(\Sigma), \supseteq\rangle$	$\alpha^{g\mathcal{W}p}(f) = \lambda P.\ \{\ s \in \Sigma \mid\ f(s) \subseteq P\ \}$
$g\mathcal{H} = \alpha^{g\mathcal{H}}(g\mathcal{W}p)$	$\langle \wp(\Sigma_\perp) \overset{\text{coa}}{\longrightarrow} \wp(\Sigma), \supseteq\rangle \underset{\alpha^{g\mathcal{H}}}{\overset{\gamma^{g\mathcal{H}}}{\leftrightarrows}} \langle \wp(\Sigma) \otimes \wp(\Sigma_\perp), \supseteq\rangle$	$\alpha^{g\mathcal{H}}(\Phi) = \{\ \langle X, Y\rangle \mid\ X \subseteq \Phi(Y)\ \}$
$\mathcal{R}^+ = \alpha^{\mathcal{R}+}(\mathcal{R}^\infty)$	$\langle \wp(\Sigma \times \Sigma_\perp), \subseteq\rangle \underset{\alpha^{\mathcal{R}+}}{\overset{\gamma^{\mathcal{R}+}}{\leftrightarrows}} \langle \wp(\Sigma \times \Sigma), \subseteq\rangle$	$\alpha^{\mathcal{R}+}(X) = X \cap (\Sigma \times \Sigma)$
$\mathcal{R}^\partial = \alpha^{\mathcal{R}\partial}(\mathcal{R}^\infty)$	$\langle \wp(\Sigma \times \Sigma_\perp), \subseteq\rangle \underset{\alpha^{\mathcal{R}\partial}}{\overset{\gamma^{\mathcal{R}\partial}}{\leftrightarrows}} \langle D^\partial, \subseteq\rangle$	$\alpha^{\mathcal{R}\partial}(X) = X \cup \left\{\ \langle \sigma_0, s\rangle \mid\ \begin{array}{l}\langle \sigma_0, \perp\rangle \in X \\ \wedge\ s \in \Sigma\end{array}\ \right\}$
$\mathcal{R}^\omega = \alpha^{\mathcal{R}\omega}(\mathcal{R}^\infty)$	$\langle \wp(\Sigma \times \Sigma_\perp), \subseteq\rangle \underset{\alpha^{\mathcal{R}\omega}}{\overset{\gamma^{\mathcal{R}\omega}}{\leftrightarrows}} \langle \wp(\Sigma \times \{\perp\}), \subseteq\rangle$	$\alpha^{\mathcal{R}\omega}(X) = X \cap (\Sigma \times \{\perp\})$

Table 1: Basic semantics and observables as abstract interpretations

components of a program and ρ is a closure operator representing an observable property of the semantics, then the corresponding semantics is compositional if

$$(\textsc{Comp})\quad \rho(X^\frown Y) = \rho(\rho(X)^\frown \rho(Y))$$

where the concatenation operator, $^\frown$, is the canonical way of composing traces. The idea is that we can compose the observations made on partial computations and obtain back, as result and without any loss of precision, the observation of the whole computation. Let's consider a well-known compositional semantics as regards the concatenation of traces, which is the maximal trace semantics. We remind that if we denote as $[\![\cdot]\!]$ this semantics, then we can describe its compositionality as $[\![P_1; P_2]\!] = [\![P_1]\!]^\frown [\![P_2]\!]$, where P_1 and P_2 are generic programs. Consider a GI: $\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq_A\rangle$ defined on the concrete domain of the maximal traces $\wp(\Sigma^\infty)$. This induces an abstract semantics which is defined on the abstract domain of denotations A: $[\![\cdot]\!]^A \stackrel{\text{def}}{=} \alpha([\![\cdot]\!])$. Our aim in this paper is that of characterizing those abstract semantics that are compositional as regards the concatenation of programs, i.e. such that $[\![P_1; P_2]\!]^A = [\![P_1]\!]^A \diamond [\![P_2]\!]^A$. In this equation we have the abstract operation $\diamond$ that has to approximate the concrete composition of traces on abstract denotations. The best correct approximation of $^\frown$ in A is by definition the best choice defining $\diamond$: $[\![P_1]\!]^A \diamond [\![P_2]\!]^A \stackrel{\text{def}}{=} \alpha(\gamma([\![P_1]\!])^\frown\gamma([\![P_2]\!]))$. Note that we have $[\![P_1; P_2]\!]^A = \alpha([\![P_1; P_2]\!]) = \alpha([\![P_1]\!]^\frown [\![P_2]\!])$. It is known that, if we consider $\diamond$ as the best correct approximation of $^\frown$ as defined above, then the relation $\alpha([\![P_1]\!]^\frown [\![P_2]\!]) \leq [\![P_1]\!]^A \diamond [\![P_2]\!]^A = \alpha(\gamma([\![P_1]\!])^\frown\gamma([\![P_2]\!]))$ holds. Now, for the relation between GI and closure operators, its clear that the abstract semantics that satisfy the equation (Comp) are exactly those semantics that make this relation an equality. This means that the equation (Comp) characterizes precisely the semantics that are compositional as regards the concatenation of traces. Clearly, not all the semantics satisfy condition (Comp), as shown in the following example.

Example 4.1. *Consider the semantics obtained by approximating the trace semantics with the abstraction function* $\alpha(X) = \{\ \sigma_{n-1} \mid\ \sigma \in X\ \wedge\ |\sigma| = n\ \}$. *This semantics observes the final states of finite computations only. Consider the program*

$$P \left[\begin{array}{l} P_1 \left[\begin{array}{l} x := 0; \\ \textbf{while } x \leq 3 \textbf{ do } x := x + 1; \end{array}\right. \\ \\ P_2 \left[\begin{array}{l} y := 0; \\ z := x + y; \end{array}\right. \end{array}\right.$$

The concrete semantics of P, *where the states are the values in* $\mathbb{N}_\perp$ *of the variables of* P, *is the only finite trace*

$$\begin{array}{rcl}[\![P]\!] & = & \{\langle 0,\perp,\perp\rangle \to \langle 1,\perp,\perp\rangle \to \langle 2,\perp,\perp\rangle \to \\ & & \to \langle 3,\perp,\perp\rangle \to \langle 4,\perp,\perp\rangle \to \langle 4,0,\perp\rangle \to \langle 4,0,4\rangle\}\end{array}$$

and the concrete semantics of P_1 *and* P_2 *are*

$$\begin{array}{rcl}[\![P_1]\!] & = & \{\langle 0,\perp,\perp\rangle \to \langle 1,\perp,\perp\rangle \to \langle 2,\perp,\perp\rangle \to \\ & & \to \langle 3,\perp,\perp\rangle \to \langle 4,\perp,\perp\rangle\} \\ [\![P_2]\!] & = & \{\langle \perp,0,\perp\rangle \to \langle \perp,0,\perp\rangle \to \langle \perp,0,\perp\rangle\}\end{array}$$

Then the abstract semantics, denoted by $[\![\cdot]\!]^A \stackrel{def}{=} \alpha([\![\cdot]\!])$, *are*

$$\begin{array}{rcl}[\![P]\!]^A & = & \{\langle 4,0,4\rangle\} \\ [\![P_1]\!]^A & = & \{\langle 4,\perp,\perp\rangle\} \\ [\![P_2]\!]^A & = & \{\langle \perp,0,\perp\rangle\}\end{array}$$

It is easy to observe that there is no way of combining $[\![P_1]\!]^A$ *and* $[\![P_2]\!]^A$ *in order to obtain* $[\![P]\!]^A$, *since we have lost the history of the computation of the programs above.*

Another problem related with equation (Comp) corresponds to characterize how the semantics ρ behaves in presence of

divergence. It is known that in standard semantics [33], if the X component diverges then $\rho(X^\frown Y) = \rho(X)$. Even if equation (Comp) holds, this fact represents a bound in compositionality because for any two computations $Y \neq Z$, if X diverges then $\rho(X^\frown Y) = \rho(X) = \rho(X^\frown Z)$. In this case we are unable to characterize what may happen beyond divergent computations. On the contrary this information is essential in order to give a consistent model for program manipulation [32]. Program slicing is a program manipulation technique that identifies syntactically well-defined portions of programs which are semantically significant but not necessarily close to each other. In particular a *program slice* is defined as a set of program statements that directly or indirectly contribute to the values assumed by a set of variables at some program point. Whenever a *program* is *sliced* we have the problem of how the semantics of a slice relates with the semantics of the original program. In general *slicing* can change the termination status, and therefore the program behavior. Indeed a slice can terminate on specific inputs that lead the original program to non-termination. Note that slicing can't introduce divergence, it can only introduce termination.

Example 4.2. *Consider the following programs written in* Imp*, taken from [26]:*

$$P_1 \left[\begin{array}{l} x := 0\,; \\ i := 1\,; \textbf{ while } i > 0 \textbf{ do } i := i + 1\,; \\ y := x\,; \end{array}\right.$$

$$P_2 \left[\begin{array}{l} x := 0\,; \\ w := 1\,; \\ y := x\,; \end{array}\right. \qquad P_3 \left[\begin{array}{l} x := 0\,; \\ \\ y := x\,; \end{array}\right.$$

Note that P_3 is a slice of both P_1 and of P_2. Let $\sqsubseteq_{sl}$ be the relation "is-slice-of"[2]*, and let $\sqsubseteq_{sem}$ be the relation of semantic approximation. In standard denotational semantics of imperative languages (denoted as $[\![\cdot]\!]$) the commands are denoted as store transformations. P_1 contains a loop so we have that $[\![P_1]\!] = \lambda s.\, \bot$. Hence $P_3 \sqsubseteq_{sl} P_1$ and $P_3 \sqsubseteq_{sl} P_2$ but it is clear that $[\![P_1]\!] \sqsubseteq_{sem} [\![P_3]\!] \sqsubseteq_{sem} [\![P_2]\!]$. In other words in the standard semantics of imperative languages the relation "is-slice-of" is inconsistent with the semantic approximation relation.*

Compositionality as described in equation (Comp) is necessary for modeling program manipulation, but it is not sufficient, as shown above for program slicing. In this example it is clear that the problem of the semantics lies upon the fact that it is unable to distinguish programs that may diverge. This means that a semantics is consistent with the program slicing if, independently from the termination status of program's sub-components, the semantics can always distinguish the meaning of programs even if some of its subcomponents diverge. We can formalize this fact in the following way:

$$[\![P_1]\!] \neq [\![P_2]\!] \;\Rightarrow\; \forall Q, W\,.\, [\![Q]\!] \diamond [\![P_1]\!] \diamond [\![W]\!] \neq [\![Q]\!] \diamond [\![P_2]\!] \diamond [\![W]\!].$$

It is clear that the above holds iff

$$[\![P_1]\!] \neq [\![P_2]\!] \;\Rightarrow\; \forall Q\,.\, [\![Q]\!] \diamond [\![P_1]\!] \neq [\![Q]\!] \diamond [\![P_2]\!].$$

[2]See [25] for a formal definition of the relation "is-slice-of" in terms of *program dependency graphs*, PDG.

In order to solve this problem, the composition $\rho(X) \diamond \rho(Y)$ of two semantics $\rho(X)$ and $\rho(Y)$ should keep trace of what happens in Y even if X diverges. This corresponds to ask a stronger form of compositionality with respect to equation (Comp), namely that it is possible to subtract part of the computation and still be able to re-compose the whole computation back by composing its fragments. This form of compositionality is called *strong compositionality*. In a certain sense we are looking for a semantics which is both compositional, as in (Comp), and monotone with respect to a relation $\tilde{\sqsubseteq}$ on semantic objects, such that $X \tilde{\sqsubseteq} Y$ iff $X = [\![P_1]\!]$, $Y = [\![P_2]\!]$, and $P_1 \sqsubseteq_{sl} P_2$. Clearly, the standard maximal trace semantics, even though compositional, is not concrete enough to provide the ground for defining semantics which are strongly compositional, i.e. monotone with respect to $\tilde{\sqsubseteq}$.

In the following we introduce systematic methods for deriving semantics of programming languages which are both compositional and strongly compositional, namely they are adequate for modeling program manipulation techniques like slicing. We provide a domain-theoretic characterization of these properties as solutions of simple recursive abstract domain equations. This will correspond to ask for a lazy compositional semantics to be compositional as specified in equation (Comp), even beyond non-terminating traces.

5. TRANSFINITE SEMANTICS

In order to find a compositional semantics which is useful for slicing we consider traces able to *look* beyond the infinite computations, namely we use semantics represented by *transfinite state traces* of programs (e.g. see [23]). For transfinite traces we mean traces whose length can be any $\beta \in \mathbb{O}$. In this way the finite computations are finite traces, while the infinite computations can have lengths which overcome the first infinite ordinal ω.

We can generalize the definition of finite trace obtaining the set of traces of length at most β, with $\beta \in \mathbb{O}$:

$$\Sigma^{<\beta} \stackrel{\text{def}}{=} \bigcup_{\alpha \in \beta} \alpha \longrightarrow \Sigma$$

With this notation it is evident that $\Sigma^+ = \Sigma^{<\omega}$. In order to specify the relation between semantics observing program's behavior at different β-depth, with $\beta \in \mathbb{O}$, as abstract interpretations of a most concrete transfinite semantics, we have to find if there exists an upper-bound, expressed in terms of number of states, on the possible length of a generic trace in the semantics of an Imp program.

Theorem 5.1. *Let P a program written in* Imp*. The upper bound for the length of possible traces expressed in term of number of states of P is $\omega^{\omega+1} \in \mathbb{O}$.*

Proof. Consider a program P written in the simple imperative language Imp. We have to prove that, independently from the number of instructions (always finite) of P, the number of states is upper bounded by $\omega^{\omega+1}$.

First of all each program has a number of instructions upper bounded by ω, because it has to be executed so it can have only a finite number of instructions. Now we have to find the possible bounds for the single instructions. The command **while** is the only one that can diverge and so it is the command that can generate the greater number

of states, if compared with all the other commands. So a generic program with n instructions has a number of possible states which is surely less than a program that has n **while**'s. Therefore suppose that the program contains only **while**'s. We have to find the upper bound for this command. Now we prove, by induction on the nesting level of **while** that if n is the maximum number of nested **while**, then the bound for the possible states of the most external **while** is ω^{2n+2}. If the nesting level is 0, then we have a **while** only with instructions with a finite number of states, so the upper bound of each instruction is ω. In the worst case the command diverges and so we have ω iterations, this means that the length is bounded by $\omega \cdot \omega = \omega^2$.
Now suppose that a **while**, with a nesting level of n, is bounded by ω^{2n+2} states. We prove that we can add another **while** externally, enhancing the level of nesting to $n+1$, then the upper bound becomes ω^{2n+4}. A generic **while** with maximum nesting level $n+1$ has surely a number of states less than the one that contains only **while** commands with maximum level n. Then the possible states for this instruction is the number of iterations, bounded by ω, multiplied by the bound of each instruction, that for inductive hypothesis is ω^{2n+2}, multiplied by the number of instructions in the **while**, bounded by ω. So we have the bound $\omega^{2n+2}\omega\omega = \omega^{2n+4}$. This means that the bound of a generic **while** is ω^ω, because we know, from [22], that $\bigcup \{\ \omega^m \mid\ m \in \omega\ \} = \omega^\omega$. Because a generic program has a number of instruction bounded by ω, the final bound is $\omega^\omega \cdot \omega = \omega^{\omega+1}$. □

We will denote this maximal ordinal as $\propto \stackrel{\text{def}}{=} \omega^{\omega+1}$. At this point we can define the maximal transfinite trace semantics as the semantics represented by the traces of the domain $\wp(\Sigma^\propto)$, where

$$\Sigma^\propto \stackrel{\text{def}}{=} \bigcup_{\alpha \in \propto} \alpha \longrightarrow \Sigma$$

Consider for instance the following program written in Imp:

$$P \left[\begin{array}{l} x := 1; \\ \textbf{while } x > 0 \textbf{ do } x := 3; \\ x := 5; \end{array} \right.$$

It is clear that the program loops infinitely but it is also clear that after the execution of **while** there exists another state of the program, which cannot be reached in any finite computation, and which associates the value 5 with the variable x. So we think of studying computations which lead to the infinite, in order to observe what happens after a divergence. We can generalize the concept of transition system to the transfinite case by supposing that the transition relation can distinguish at which degree of infinite the transition is possible. Let the *length* of a transfinite trace be $\alpha \in \mathbb{O}$. If α is a limit ordinal then the index of the last state is α, otherwise it is $\alpha - 1$. In general we will denote always with $\sigma_{\kappa-1}$ the last state of a transfinite trace.

In order to generalize the trace concatenation to transfinite traces, we have to specify a topology on transfinite traces which allows us to specify limit states for traces whose length is a limit ordinal. This is achieved by considering the standard topology on traces as induced by $\mathbb{O}$: The limit ordinal are the *open* elements on $\mathbb{O}$ while the successor ones are the *closed* elements of the topology [23]. This induces a topology on $\Sigma^\propto$ and in the following we assume that this topology is metrizable and complete [31], i.e. any Cauchy sequence of transfinite traces has a limit in $\Sigma^\propto$.

The transfinite semantics is more concrete than the maximal trace semantics defined in the Cousot semantic hierarchy. In the following we denote the semantics of transfinite traces of a program by $\tau^\propto$.

Example 5.2. *Consider the following program written in* Imp*:*

$$P \left[\begin{array}{l} x := 2; \\ \textbf{while } x > 0 \textbf{ do } x := x + 1; \\ x := 5; \end{array} \right.$$

we represent the trace of states on the maximal trace semantics by considering the trace of the values of x in the computation:

$$2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow\ \cdots\ \longrightarrow n \longrightarrow\ \cdots\ \bot$$

Namely the maximal trace semantics is not able to see beyond the loop. Instead if we consider the transfinite trace semantics, the trace becomes

$$2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow\ \cdots\ \longrightarrow n \longrightarrow\ \cdots\ \omega\ \longrightarrow 5$$

In the following we prove that the maximal trace semantics τ^∞ is indeed an abstract interpretation of the transfinite semantics $\tau^\propto$ given by a Galois insertion between $\wp(\Sigma^\infty)$ and $\wp(\Sigma^\propto)$. We define the functions $\alpha^\propto : \wp(\Sigma^\propto) \longrightarrow \wp(\Sigma^\infty)$ and $\gamma^\propto : \wp(\Sigma^\infty) \longrightarrow \wp(\Sigma^\propto)$ as

$$\alpha^\propto(X) = \{\ \sigma \in \Sigma^\omega \mid\ \exists \delta \in X\ .\ \sigma \preccurlyeq \delta\ \} \cup (X \cap \Sigma^+)$$
$$\gamma^\propto(Y) = (Y \cap \Sigma^+) \cup \{\ \sigma \in \Sigma^\propto \mid\ \exists \delta \in Y^\omega\ .\ \delta \preccurlyeq \sigma\ \}$$

The relation $\preccurlyeq$ is the prefix relation generalized to transfinite traces, namely if we define the relation $\sigma \preccurlyeq^n \delta$ as σ cut at the length n is prefix of δ, then we can define $\sigma \preccurlyeq^\infty \delta$ as $\forall n \in \omega\ .\ \sigma \preccurlyeq^n \delta$. This is the same as $\preccurlyeq^\infty \stackrel{\text{def}}{=} \bigcap_{n \in \omega} \preccurlyeq^n$. In the following we will denote with $\preccurlyeq$ the relation $\preccurlyeq^\infty$.

Proposition 5.3.

$$\langle \wp(\Sigma^\propto), \subseteq \rangle \underset{\alpha^\propto}{\overset{\gamma^\propto}{\leftrightarrows}} \langle \wp(\Sigma^\infty), \subseteq \rangle$$

The following theorem proves that the standard finite and infinite maximal trace semantics is an abstract interpretation of the non-standard transfinite one.

Theorem 5.4. $\tau^\infty = \alpha^\propto(\tau^\propto)$.

6. SEMANTICS BY REDUCED POWER

In this section we systematically construct compositional (resp. denotational and weakest precondition) semantics by composing more abstract non-compositional semantics, and we prove that these semantics are optimal, viz. most abstract with respect to the domain refinement operation given by reduced power. This proves that these semantics can be obtained as the most abstract solution of an abstract domain equation which involves the reduced power and a basic non-compositional semantics. Kleene algebras are the typical algebraic structures to reason about traces [5, 12]. A *Kleene algebra* is a sixtuple $\langle K, \leq, \top, \cdot, \bot, 1 \rangle$ such that $\langle K, \leq, \vee, \wedge, \top, \bot \rangle$ is a complete lattice, $\langle K, \cdot, 1 \rangle$ is a monoid and $\cdot : K \times K \longrightarrow K$ is both left and right additive. In order to apply the reduced relative power operation to design compositional semantics out of non-compositional ones, we need a quantale structure on transfinite traces. Kleene algebras provide this structure.

PROPOSITION 6.1. *$\langle \wp(\Sigma^\infty), \subseteq, \Sigma^\infty, ^\frown, \varnothing, \Sigma\rangle$ is a Kleene algebra providing a unitary quantale, where for any $X, Y \in \wp(\Sigma^\infty)$ we have $X^\frown Y \stackrel{def}{=} \{\ \sigma\delta \mid \sigma \in X \ \wedge\ \sigma_{\kappa-1}\delta \in Y\ \}$.*

6.1 Potential λ-termination semantics

In Example 4.1 we have shown a non-compositional semantics observing final terminating states. We generalize that semantics to transfinite traces.

We know that if $\alpha \in \mathbb{O}$ then there exist (unique) $\lambda \in \mathbb{O}_l$ and $n \in \omega$ such that $\alpha = \lambda + n$. This corresponds to say that $\lambda \leq \alpha < S_l(\lambda)$, where $S_l(\lambda)$ is the limit ordinal which follows λ. We also note that the limit ordinal λ is such that $\forall \gamma \in \mathbb{O}_l\,.\,\gamma \leq \alpha$ and $\gamma \leq \lambda$, namely such λ is the biggest limit ordinal contained in α. In the following we denote by $\langle s, \lambda\rangle$ with $s \in \Sigma$ and $\lambda \in \mathbb{O}_l$ the state s, in a possibly transfinite trace σ, located, in σ, in a position which is after the limit ordinal λ. Hence, $\langle \sigma_{\kappa-1}, \lambda\rangle$ denotes the final state of a possibly transfinite trace whose length overcomes λ and $\langle \sigma_0, \lambda\rangle$ is the initial state of the sub-trace of σ starting after λ. Clearly, the initial and final states in a finite trace $\sigma \in \Sigma^+$ are respectively $\langle \sigma_0, 0\rangle$ and $\langle \sigma_{\kappa-1}, 0\rangle$. This notation allows us to keep trace of states between any two consecutive limit ordinals, namely states that can be located after any sequence of divergent computations.
Therefore we define the two functions $\alpha^\vdash : \wp(\Sigma^\infty) \longrightarrow \wp(\Sigma)$ and $\gamma^\vdash : \wp(\Sigma) \longrightarrow \wp(\Sigma^\infty)$, where we abbreviate with Σ the set $\Sigma\times\infty$:

$$\alpha^\vdash(X) = \{\ \langle \sigma_{\kappa-1}, \lambda\rangle \mid \sigma \in X \ \wedge\ \lambda \leq |\sigma| < S_l(\lambda)\ \}$$
$$\gamma^\vdash(Y) = \{\ \sigma \in \Sigma^\infty \mid \langle \sigma_{\kappa-1}, \lambda\rangle \in Y \ \wedge\ \lambda \leq |\sigma| < S_l(\lambda)\ \}$$

Dually we can define the maps $\alpha^\dashv : \wp(\Sigma^\infty) \longrightarrow \wp(\Sigma)$ and $\gamma^\dashv : \wp(\Sigma) \longrightarrow \wp(\Sigma^\infty)$ as follows

$$\alpha^\dashv(X) = \{\ \langle \sigma_0, \lambda\rangle \mid \sigma \in X \ \wedge\ \lambda \leq |\sigma| < S_l(\lambda)\ \}$$
$$\gamma^\dashv(Y) = \{\ \sigma \in \Sigma^\infty \mid \langle \sigma_0, \lambda\rangle \in Y \ \wedge\ \lambda \leq |\sigma| < S_l(\lambda)\ \}$$

PROPOSITION 6.2. *Under the previous hypothesis:*

- $\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^\vdash}{\overset{\gamma^\vdash}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq\rangle$
- $\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^\dashv}{\overset{\gamma^\dashv}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq\rangle$

The closure operators associated with the, respectively so called, *forward* and *backward λ-termination semantics* are the following functions:

$$\rho^\vdash(X) \stackrel{\text{def}}{=} \gamma^\vdash\alpha^\vdash(X) = \left\{\ \sigma \in \Sigma^\infty \;\middle|\; \begin{array}{l} \exists \delta \in X\,.\,\sigma_{\kappa-1} = \delta_{\kappa-1} \\ \lambda \leq |\sigma|, |\delta| < S_l(\lambda) \end{array}\ \right\}$$
$$\rho^\dashv(X) \stackrel{\text{def}}{=} \gamma^\dashv\alpha^\dashv(X) = \left\{\ \sigma \in \Sigma^\infty \;\middle|\; \begin{array}{l} \exists \delta \in X\,.\,\sigma_0 = \delta_0 \ \wedge \\ \lambda \leq |\sigma|, |\delta| < S_l(\lambda) \end{array}\ \right\}$$

where the ordinals are all limit and they are the biggest ones less than the length of the considered traces. We denote with $\tau^\vdash$ and $\tau^\dashv$ respectively the domains $\rho^\vdash(\tau^\infty)$ and $\rho^\dashv(\tau^\infty)$. Note that $\tau^\dashv$ generalizes the potential termination semantics in [8], here defined for finite traces only.

6.2 Systematic construction of $\mathcal{D}$ semantics

In this section we systematically derive the denotational semantics $\mathcal{D}$ by refining the potential termination semantics with respect to the input/output relations. This is obtained by considering as refined domain the space of all monotone functions between sets of states. The idea is to approximate transfinite traces with input/output relations, which are functions in the case of denotational semantics. In this section we prove that this construction can be obtained as the backward reduced power of the potential forward λ-termination semantics. In this case the abstraction function is given by construction: $\alpha^D(X) = \lambda Y.\, \alpha^\vdash(\gamma^\vdash(Y)^\frown X)$.

PROPOSITION 6.3.

$$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^D}{\overset{\gamma^D}{\leftrightarrows}} \langle \wp(\Sigma) \longrightarrow \wp(\Sigma), \sqsubseteq\rangle$$

where:

$$\alpha^D(X) = \lambda Y.\ \left\{\ \langle \eta_{\kappa-1}, \lambda_1\rangle \;\middle|\; \begin{array}{l} \eta \in X \ \wedge\ \langle \eta_0, \lambda\rangle \in Y \\ \wedge\ \lambda_1 \leq \lambda + |\eta| < S_l(\lambda_1) \end{array}\ \right\}$$
$$\gamma^D(f) = \left\{\ \sigma \in \Sigma^\infty \;\middle|\; \begin{array}{l} \exists \lambda\,.\,\langle \sigma_{\kappa-1}, \lambda'\rangle \in f(\langle \sigma_0, \lambda\rangle) \\ \lambda' \leq \lambda + |\sigma| < S_l(\lambda') \end{array}\ \right\}$$

The semantics obtained so far by reduced power construction models the space of functions on $\wp(\Sigma)$ representing input/output relations of states in transfinite traces. We denote by $\mathcal{D}^\infty \stackrel{\text{def}}{=} \alpha^D(\tau^\infty)$. The following theorem proves that the denotational semantics in [8] corresponds precisely to backward reduced power of potential forward λ-termination semantics, by observing input/output functionalities of traces whose length is at most ω.

THEOREM 6.4. $\mathcal{D} = \alpha^{\mathcal{D}} \circ \alpha^{\mathcal{R}} \circ \alpha^\infty \circ \gamma^D(\mathcal{D}^\infty)$.

6.3 Systematic construction of $g\mathcal{W}p$ semantics

The weakest precondition semantics can be obtained in a similar way by dualizing the construction of the denotational semantics. Duality here means exchanging the observation from the final to the initial state and reversing the relative power operation. In this section we derive the transfinite weakest precondition semantics as the forward reduced power of the potential backward λ-termination semantics. As before, the corresponding abstraction function follows by construction. Let $X \subseteq \Sigma^\infty$, then by construction: $\alpha^W(X) = \lambda Y.\, \alpha^\dashv(X^\frown\gamma^\dashv(Y))$.

PROPOSITION 6.5.

$$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^W}{\overset{\gamma^W}{\leftrightarrows}} \langle \wp(\Sigma) \longrightarrow \wp(\Sigma), \sqsubseteq\rangle$$

where:

$$\alpha^W(X) = \lambda Y.\ \left\{\ \langle \eta_0, \lambda_1\rangle \;\middle|\; \begin{array}{l} \eta \in X \ \wedge\ \langle \eta_{\kappa-1}, \lambda\rangle \in Y \\ \wedge\ \lambda_1 \leq \lambda + |\eta| < S_l(\lambda_1) \end{array}\ \right\}$$
$$\gamma^W(f) = \left\{\ \sigma \in \Sigma^\infty \;\middle|\; \begin{array}{l} \exists \lambda\,.\,\langle \sigma_0, \lambda'\rangle \in f(\langle \sigma_{\kappa-1}, \lambda\rangle) \\ \lambda' \leq \lambda + |\sigma| < S_l(\lambda') \end{array}\ \right\}$$

This semantics models the space of functions on $\wp(\Sigma)$ representing the largest set of states that may lead the possibly transfinite computation in a given set of output states. We denote by $\mathcal{W}p^\infty \stackrel{\text{def}}{=} \alpha^W(\tau^\infty)$. The following theorem proves that the weakest precondition semantics defined in [8] corresponds precisely to reduced power of potential backward λ-termination semantics, by observing the largest set of states that reach a given output state in at most ω steps.

THEOREM 6.6. $g\mathcal{W}p = \alpha^{g\mathcal{W}p} \circ \alpha^{\mathcal{D}} \circ \alpha^{\mathcal{R}} \circ \alpha^\infty \circ \gamma^W(\mathcal{W}p^\infty)$.

In Fig. 2 we show the transfinite semantics, here called *non-standard semantics*, with respect to Cousot's hierarchy

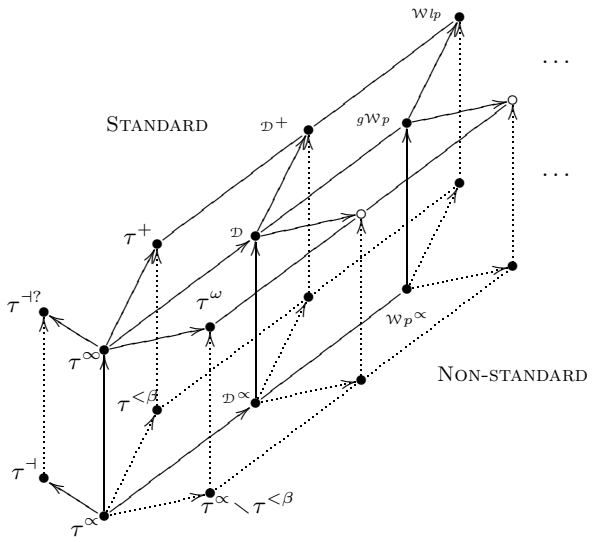

Figure 2: Symmetrical standard and non-standard semantics.

of standard semantics. The standard natural semantics is here the "projection", specified as an abstraction, of the natural transfinite level through the pair of adjoint functions $\alpha^\propto$ and $\gamma^\propto$. With each natural transfinite semantics $\tau^\propto$, $\mathcal{D}^\propto$, and $\mathcal{W}p^\propto$ and a given ordinal $\beta \in \mathbb{O}$, it is possible to specify symmetric approximate semantics observing computations up to β ($\tau^{<\beta}$ etc.) and observing computations longer than β ($\tau^{<\propto} \smallsetminus \tau^{<\beta}$ etc.). It is easy to observe that the standard angelic and infinite semantics can be obtained by abstract interpretation and in particular:

$$\tau^{<\beta} = \gamma^{<\beta} \circ \alpha^{<\beta}(\tau^\propto) \quad \text{and} \quad \tau^\propto \smallsetminus \tau^{<\beta} = \gamma^{\geq\beta} \circ \alpha^{\geq\beta}(\tau^\propto)$$

where

$$\begin{array}{ll} \alpha^{<\beta}(X) = X \cap \Sigma^{<\beta} & \alpha^{\geq\beta}(X) = X \cap (\Sigma^\propto \smallsetminus \Sigma^{<\beta}) \\ \gamma^{<\beta}(Y) = Y \cup (\Sigma^\propto \smallsetminus \Sigma^{<\beta}) & \gamma^{\geq\beta}(Y) = Y \cup \Sigma^{<\beta}. \end{array}$$

Note that if $\beta = \omega$, then $\tau^{<\omega} = \tau^+$ and $\alpha^\propto(\tau^\propto \smallsetminus \tau^{<\omega}) = \tau^\omega$. The remaining abstractions from the non-standard to the standard hierarchy in Fig. 2, can be derived in a similar way for all the semantics.

The equivalence between denotational and weakest precondition semantics proved in [8] can be generalized to the transfinite case. This follows directly by construction and can be emphasized by considering the corresponding closure operators. It is immediate to observe that the standard denotational semantics viewed as a closure operation on maximal traces is:

$$Den(X) = gWp(X) = \left\{ \sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} \exists \delta \in X^+ \,.\, \delta_0 = \sigma_0 \\ \wedge\ \delta_{n-1} = \sigma_{n-1} \end{array} \right\} \cup \left\{ \sigma \in \Sigma^\omega \;\middle|\; \exists \delta \in X \smallsetminus \Sigma^+ \,.\, \sigma_0 = \delta_0 \right\}$$

As a consequence of Theorem 6.4 and 6.6, this semantics can be obtained by applying the abstraction $\alpha^\propto$ to the domains $\tau^\vdash \overleftarrow{\frown} \tau^\vdash$ and $\tau^\dashv \overrightarrow{\frown} \tau^\dashv$, where $\gamma^D \circ \alpha^D \stackrel{\text{def}}{=} \tau^\vdash \overleftarrow{\frown} \tau^\vdash$ and $\gamma^W \circ \alpha^W \stackrel{\text{def}}{=} \tau^\dashv \overrightarrow{\frown} \tau^\dashv$.

PROPOSITION 6.7.

$$\tau^\vdash \overleftarrow{\frown} \tau^\vdash = \tau^\dashv \overrightarrow{\frown} \tau^\dashv = \lambda X. \left\{ \sigma \in \Sigma^\propto \;\middle|\; \begin{array}{l} \exists \eta \in X \,.\, \eta_{\kappa-1} = \sigma_{\kappa-1} \\ \eta_0 = \sigma_0 \\ \exists \lambda' \,.\, \lambda' \leq |\eta|, |\sigma| < S_l(\lambda') \end{array} \right\}$$

COROLLARY 6.8.

$$Den = gWp = \alpha^\propto \circ (\tau^\vdash \overleftarrow{\frown} \tau^\vdash) = \alpha^\propto \circ (\tau^\dashv \overrightarrow{\frown} \tau^\dashv).$$

REMARK 6.9. *It can be easily verified that, by inverting the direction of the arrow in $\tau^\vdash \overleftarrow{\frown} \tau^\vdash$, we obtain the identity, namely it is immediate to prove that $\tau^\vdash \overrightarrow{\frown} \tau^\vdash = \iota_{\wp(\Sigma)}$. Intuitively this happens because the forward reduced power of the forward λ-termination semantics encodes how a given set X of concrete traces behaves when these are extended with any possible trace ending in a given set of observable states Y. Hence by observing the final states of these extended traces we get back Y. Instead, if we consider the initial states in this construction we can observe the set of initial states of concrete traces that will have final states in Y. This is precisely Dijkstra's weakest precondition semantics gWp, as proved in Theorem 6.6. A similar reasoning holds if we dualize $\tau^\dashv \overrightarrow{\frown} \tau^\dashv$.*

6.4 Optimal natural denotational semantics

Now we can prove that the transfinite denotational and weakest precondition semantics obtained so far are respectively the most abstract solutions of the abstract domain equations:

$$X = \tau^\vdash \sqcap (X \overleftarrow{\frown} X) \text{ and } X = \tau^\dashv \sqcap (X \overrightarrow{\frown} X)$$

This allows us to prove an optimality result for $\tau^\vdash \overleftarrow{\frown} \tau^\vdash$ and $\tau^\dashv \overrightarrow{\frown} \tau^\dashv$ with respect to the capability of observing input/output relations of transfinite traces, namely we prove that these semantics are the most abstract ones observing $\tau^\vdash$ (resp. $\tau^\dashv$) and which are closed under $\overleftarrow{\frown}$ (resp. $\overrightarrow{\frown}$).

THEOREM 6.10.

- $(\tau^\vdash \overleftarrow{\frown} \tau^\vdash) \overleftarrow{\frown} (\tau^\vdash \overleftarrow{\frown} \tau^\vdash) = \tau^\vdash \overleftarrow{\frown} \tau^\vdash$
- $(\tau^\dashv \overrightarrow{\frown} \tau^\dashv) \overrightarrow{\frown} (\tau^\dashv \overrightarrow{\frown} \tau^\dashv) = \tau^\dashv \overrightarrow{\frown} \tau^\dashv$

We can conclude that the domain $\tau^\vdash \overleftarrow{\frown} \tau^\vdash$ is the most abstract solution of the equation $X = \tau^\vdash \sqcap X \overleftarrow{\frown} X$ because it is immediate that $\tau^\vdash \sqsubseteq \tau^\vdash \overleftarrow{\frown} \tau^\vdash$. A similar result holds for $\tau^\dashv \overrightarrow{\frown} \tau^\dashv$ with respect to $X = \tau^\dashv \sqcap (X \overrightarrow{\frown} X)$.

7. ANGELIC SEMANTICS

In this section we follow the construction given above in order to systematically derive the angelic denotational and Dijkstra's weakest liberal precondition semantics by the forward (resp. backward) reduced relative power of the backward potential 0-termination semantics (resp. the forward potential 0-termination semantics). It is worth noting that the backward potential 0-termination semantics is the *potential termination semantics* in [8]. The forward potential 0-termination semantics observes the final states of finite traces. This semantics is built as abstraction of the natural trace semantics, by using the functions $\alpha^{\vdash?} : \wp(\Sigma^\infty) \to \wp(\Sigma)$ and $\gamma^{\vdash?} : \wp(\Sigma) \to \wp(\Sigma^\infty)$ defined as follows:

$$\begin{array}{l} \alpha^{\vdash?}(X) = \left\{ \sigma_{n-1} \;\middle|\; \sigma \in X^+ \right\} \\ \gamma^{\vdash?}(Y) = \left\{ \sigma \in \Sigma^+ \;\middle|\; \sigma_{n-1} \in Y \right\} \cup \Sigma^\omega \end{array}$$

and constitute a Galois insertion. The corresponding closure is given as follows:

$$\rho^{\vdash?}(X) \stackrel{\text{def}}{=} \gamma^{\vdash?}\alpha^{\vdash?}(X) = \left\{ \sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} \exists \delta \in X^+ \,. \\ \delta_{n-1} = \sigma_{n-1} \end{array} \right\} \cup \Sigma^\omega$$

In the following we will denote $\tau^{\vdash?} \stackrel{\text{def}}{=} \rho^{\vdash?}(\tau^\infty)$.

Let $\alpha^{D^+} : \wp(\Sigma^\infty) \longrightarrow (\wp(\Sigma) \longrightarrow \wp(\Sigma))$ be the function build by reduced power as $\alpha^{D^+}(X) = \lambda Y.\, \alpha^{\vdash?}(\gamma^{\vdash?}(Y) \frown X)$ which is $\lambda Y.\, \left\{ \sigma_{n-1} \,\middle|\, \sigma \in X^+ \;\wedge\; \sigma_0 \in Y \right\}$.

LEMMA 7.1. $\mathcal{D}^+ \cong \alpha^{D^+}(\tau^\infty)$.

We can characterize the angelic denotational semantics as upper closure operator on the domain of finite and infinite traces (see [8]):

$$Ang^{\mathcal{D}}(X) = \left\{ \sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} \exists \delta \in X^+ \,.\, \sigma_0 = \delta_0 \\ \wedge\; \sigma_{n-1} = \delta_{n-1} \end{array} \right\} \cup \Sigma^\omega$$

THEOREM 7.2. $Ang^{\mathcal{D}} = \tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}$.

Similar results can be obtained for the weakest-liberal precondition semantics by means of Cousot's backward potential 0-termination semantics, i.e. $Wlp = \tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}$, where $\tau^{\dashv?}$ is the backward potential 0-termination semantics [8]. Due to the equivalence of angelic denotational and weakest-liberal precondition semantics, we have:

$$\tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?} = \tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}$$

Similar results for optimality can be proved also in the angelic case.

THEOREM 7.3.

- $(\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}) \stackrel{\frown}{\longleftarrow} (\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}) = \tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}$
- $(\tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}) \stackrel{\frown}{\longrightarrow} (\tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}) = \tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}$

This fact says us that the denotational angelic (resp. weakest liberal precondition) semantics is the fixed point of a refining process starting from $\tau^{\vdash?}$ (resp. $\tau^{\dashv?}$) by using $\stackrel{\frown}{\longleftarrow}$ (resp. $\stackrel{\frown}{\longrightarrow}$). So this semantics is the most abstract one which observe the final states of finite traces and which is closed as regards the functional relations between these states.

REMARK 7.4. *We wonder if* $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?} = \tau^{\vdash?} \sqcap \tau^{\dashv?}$ *namely, if* $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}$ *is the most abstract closure operator which observe both the input and output states of finite traces. It is easy to verify that* $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?} \neq \tau^{\vdash?} \sqcap \tau^{\dashv?}$ *and in particular:*

$$\begin{array}{rcl} \tau^{\vdash?} \sqcap \tau^{\dashv?} & = & (\rho^{\vdash?} \sqcap \rho^{\dashv?})(X) = \\ & = & \left\{ \sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} \exists \delta, \eta \in X \,.\, \sigma_0 = \delta_0 \\ \wedge\; \sigma_{n-1} = \eta_{n-1} \end{array} \right\} \cup \Sigma^\omega \end{array}$$

In this semantics we can note that the relation between input and output of traces is not expressed. Indeed there are also traces that don't necessarily have the initial and final state bounded by the fact that they belong to the same trace. In $\tau^{\vdash?} \sqcap \tau^{\dashv?}$ *we take the product of all the possible initial states with all the possible final states of traces in* X*. It is clear that* $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?} \sqsubseteq \tau^{\vdash?} \sqcap \tau^{\dashv?}$ *namely that* $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?}$ *is not the most abstract semantics more concrete then both of* $\tau^{\vdash?}$ *and of* $\tau^{\dashv?}$*.*

8. COMPOSITIONALITY

The aim of this section is that of proving that both the denotational semantics and the weakest precondition semantics, seen as closures on the transfinite trace semantics, satisfy the compositionality relation, namely they are both solutions of the equation (COMP). Moreover we will see that these semantics are optimal, namely they are the most abstract semantics on $\wp(\Sigma^\infty)$ for which this property holds. For proving this fact we will use a result in [20].

THEOREM 8.1. *The most abstract solution on* $uco(C)$ *of the equation* $\rho(X \frown Y) = \rho(\rho(X) \frown \rho(Y))$ *is*

$$\rho = \rho \sqcap (\rho \stackrel{\frown}{\longrightarrow} \iota_C) \sqcap (\iota_C \stackrel{\frown}{\longleftarrow} \rho) \sqcap ((\iota_C \stackrel{\frown}{\longrightarrow} \rho) \stackrel{\frown}{\longleftarrow} \iota_C).$$

We prove that the closure $\tau^{\vdash} \stackrel{\frown}{\longleftarrow} \tau^{\vdash} = \tau^{\dashv} \stackrel{\frown}{\longrightarrow} \tau^{\dashv}$ is the most abstract compositional semantics definable on the set of transfinite traces, which includes respectively $\tau^{\vdash}$ and $\tau^{\dashv}$ as an abstract interpretation. In the following we will denote simply with ι the closure $\iota_{\wp(\Sigma^\infty)}$, identity on $\wp(\Sigma^\infty)$.

LEMMA 8.2. *Let* $\rho \stackrel{def}{=} \tau^{\vdash} \stackrel{\frown}{\longleftarrow} \tau^{\vdash}$*. Then* $\iota \stackrel{\frown}{\longrightarrow} \rho = \rho = \rho \stackrel{\frown}{\longleftarrow} \iota$.

Hence, we can conclude that the desired result holds trivially.

THEOREM 8.3. *The closure* $\rho \stackrel{def}{=} \tau^{\vdash} \stackrel{\frown}{\longleftarrow} \tau^{\vdash}$ *is the most abstract one defined on the domain* $\wp(\Sigma^\infty)$*, solution of the equation* $\rho(X \frown Y) = \rho(\rho(X) \frown \rho(Y))$.

The same result holds for the angelic semantics. In this case we have:

LEMMA 8.4. *Let* $\rho \stackrel{def}{=} \tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?} = \tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}$*. Then* $\iota \stackrel{\frown}{\longrightarrow} \rho = \rho = \rho \stackrel{\frown}{\longleftarrow} \iota$.

The closure $\tau^{\vdash?} \stackrel{\frown}{\longleftarrow} \tau^{\vdash?} = \tau^{\dashv?} \stackrel{\frown}{\longrightarrow} \tau^{\dashv?}$ is therefore the most abstract semantics defined on the domain $\wp(\Sigma^\infty)$, which is solution of the equation (COMP).

The fact that the equation (COMP) has solution on the transfinite semantics says that we can slice programs without loosing informations about the semantics and without loosing consistency between the semantics of the hole program and the semantics of its parts. Hence the fact that the transfinite denotational semantics satisfies the equation

$$\rho(X \frown Y) = \rho(\rho(X) \frown \rho(Y))$$

says that this semantics is consistent with the relation $\sqsubseteq_{sl}$ described in Example 4.2 and therefore it is adequate for program slicing.

EXAMPLE 8.5. *Consider the programs defined in Example 4.2. Now we can define the transfinite denotational semantics which observe the final state even after infinite computations, denoted as* $[\![\cdot]\!]$.

$$[\![P_1]\!] : \begin{array}{l} x \mapsto 0 \\ i \mapsto \omega \\ y \mapsto 0 \end{array} \qquad [\![P_2]\!] : \begin{array}{l} x \mapsto 0 \\ w \mapsto 1 \\ y \mapsto 0 \end{array} \qquad [\![P_3]\!] : \begin{array}{l} x \mapsto 0 \\ y \mapsto 0 \end{array}$$

In this way it is clear that the relations between the programs semantics are:

$$[\![P_3]\!] \sqsubseteq_{sem} [\![P_1]\!] \quad \textit{and} \quad [\![P_3]\!] \sqsubseteq_{sem} [\![P_2]\!]$$

exactly as we wanted, namely they are consistent with the existing "is-slice-of" relation between programs $\sqsubseteq_{sl}$*, described in Example 4.2.*

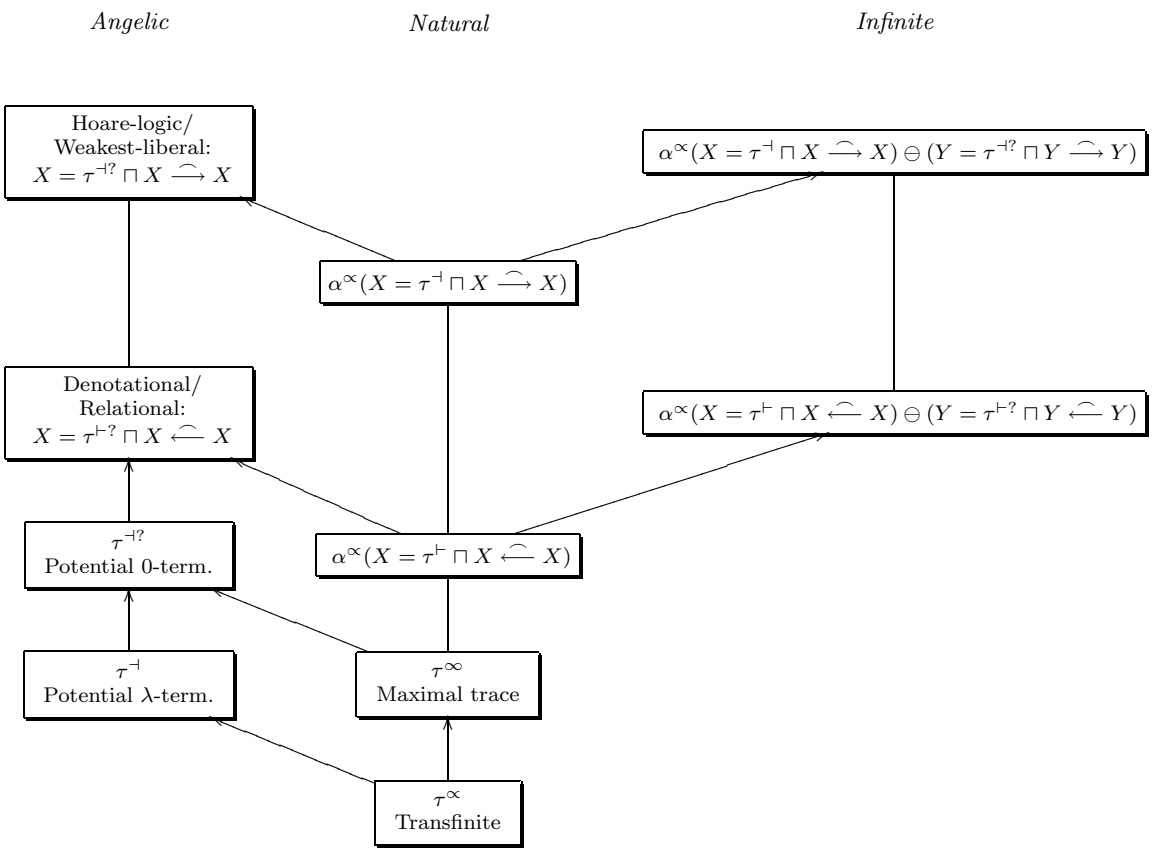

Figure 3: Semantics as abstract domain equations

9. CONCLUSION

We have shown a strong connection between the structure of relational abstract domains for program analysis and compositionality of the underlying semantics. Both can be systematically designed by solving the same abstract domain equation by means of the same domain refinement: the reduced power operation (see [28] for an example of relational abstract domains for program analysis as solutions of recursive domain equations). These results provide an equational presentation of semantics and abstract domains for program analysis in a unique formal setting. Fig. 3 shows the structure of Cousot's hierarchy of semantics as a hierarchy of abstract domain equations. Here $\ominus$ is abstract domain complementation [6], and it is applied to characterize the complementary nature of angelic and infinite semantics as shown in [16]. Moreover both finite and transfinite compositional semantics share the same structure, and this allows us to provide consistent models for program manipulation. It is worth noting that the algebraic properties of program integration techniques (and therefore of program slicing) are similar to the properties of reduced power which, as we proved in this paper, is the basic operation to refine any non-compositional semantics in order to make it compositional. Therefore similar algebraic operators can be used to model both syntactic manipulation techniques and the corresponding compositional semantics. As observed in [25], well-known algorithms for program integration, like the Horowitz, Prins, and Reps's algorithm (HPR), can be formulated in terms of intuitionistic implication on the complete Heyting algebra constructed on dependency graphs. On the other side the reduced relative power operation exploits linear implications on a given quantale [20], which is a generalization of complete Heyting algebras [21, 27]. The analogy and the relation between these two constructions deserves further research.

10. REFERENCES

[1] Ballance, R. A., Maccabe, A. B., and Ottenstein, K. J. The program dependence web: A representation supporting control-,data-, and demand-driven interpretation of imperative languages. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation* (1990), pp. 257–271.

[2] Bossi, A., Gabbrielli, M., Levi, G., and Meo, M. C. A compositional semantics for logic programs. *Theor. Comput. Sci. 122*, 1-2 (1994), 3–47.

[3] Cartwrite, R., and Felleisen, M. The semantics of program dependence. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation* (1989), pp. 13–27.

[4] Comini, M., and Levi, G. An algebraic theory of observables. In *Proc. of the 1994 Internat. Logic Programming Symp. (*ILPS '94*)* (1994), M. Bruynooghe, Ed., The MIT Press, Cambridge, Mass., pp. 172–186.

[5] Conway, J. H. *Regular algebra and finite machines*. Chapman and Hall, 1971.

[6] Cortesi, A., Filé, G., Giacobazzi, R., Palamidessi, C., and Ranzato, F. Complementation in abstract interpretation. *ACM Trans. Program. Lang. Syst. 19*, 1 (1997), 7–47.

[7] Cousot, P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation (Invited Paper). In *Proc. of the 13th Internat. Symp. on Mathematical Foundations of Programming Semantics (*MFPS '97*)* (1997), S. Brookes and M. Mislove, Eds., vol. 6 of *Electronic Notes in Theoretical Computer Science*, Elsevier, Amsterdam. URL: http://www.elsevier.nl/locate/entcs/volume6.html.

[8] Cousot, P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.* (2000). To appear, extended version of [7].

[9] Cousot, P., and Cousot, R. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM Symp. on Principles of Programming Languages (*POPL '79*)* (1979), ACM Press, New York, pp. 269–282.

[10] Cousot, P., and Cousot, R. Abstract interpretation and application to logic programs. *J. Logic Program. 13*, 2-3 (1992), 103–179.

[11] Cousot, P., and Cousot, R. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th ACM Symp. on Principles of Programming Languages (*POPL '92*)* (1992), ACM Press, New York, pp. 83–94.

[12] Desharnais, J., Möller, B., and Tchier, F. Kleene under a demonic star. In *Proc. of the 9th Internat. Conf. on Algebraic Methodology and Software Technology (*AMAST '00*)* (2000), vol. 1816 of *Lecture Notes in Computer Science*, pp. 355–370.

[13] Dijkstra, E. Guarded commands, nondeterminism and formal derivation of programs. *Comm. of The ACM 18*, 8 (1975), 453–457.

[14] Gaifman, H., and Shapiro, E. Fully abstract compositional semantics for logic programs. In *Conference Record of the 16th ACM Symp. on Principles of Programming Languages (*POPL '89*)* (1989), ACM Press, New York, pp. 134–142.

[15] Giacobazzi, R. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In *Proc. of the 13th Internat. Symp. on Theoretical Aspects of Computer Science (*STACS '96*)* (1996), C. Puech and R. Reischuk, Eds., vol. 1046 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 503–514.

[16] Giacobazzi, R., and Mastroeni, I. A characterization of symmetric semantics by domain complementation. In *Proc. of the 2nd international conference in principles and practice of declarative programming* PPDP'00 (2000), ACM press, pp. 115–126.

[17] Giacobazzi, R., and Ranzato, F. Complementing logic program semantics. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming (*ALP '96*)* (1996), M. Hanus and M. Rodríguez Artalejo, Eds., Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 238–253.

[18] Giacobazzi, R., and Ranzato, F. Refining and compressing abstract domains. In *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming (*ICALP '97*)* (1997), P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds., vol. 1256 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 771–781.

[19] Giacobazzi, R., and Ranzato, F. The reduced relative power operation on abstract domains. *Theor. Comput. Sci 216* (1999), 159–211.

[20] Giacobazzi, R., Ranzato, F., and Scozzari, F. Building complete abstract interpretations in a linear logic-based setting. In *5th International Symp. SAS* (1998), G. Levi, Ed., vol. 1503, pp. 215–229.

[21] Giacobazzi, R., and Scozzari, F. A logical model for relational abstract domains. *ACM Trans. Program. Lang. Syst. 20*, 5 (1998), 1067–1109.

[22] Just, W., and Weese, M. *Discovering modern set theory. I: The basics*, vol. 8 of *Graduate studies in mathematics*. American mathematical society, 1996.

[23] Kennaway, J. R., Klop, J. W., Sleep, M. R., and Vries, F. J. Transfinite reductions in orthogonal term rewriting systems. *Information and computation 119*, 1 (1995), 18–38.

[24] Nielson, F. Tensor products generalize the relational data flow analysis method. In *Proc. of the 4th Hungarian Computer Science Conf.* (1985), M. Arató, I. Kátai, and L. Varga, Eds., pp. 211–225.

[25] Reps, T. Algebraic properties of program integration. *Sci. Comput. Program. 17* (1991), 139–215.

[26] Reps, T., and Turnidge, T. Program specialization via program slicing. In *Proceedings of the Dagstuhl seminar on Partial evaluation* (1996), O. Danvy, R. Glueck, and P. Thiemann, Eds., Springer-Verlag, pp. 409–429.

[27] Rosenthal, K. I. *Quantales and their Applications*. Pitman Research Notes in Mathematics. Longman Scientific & Technical, London, 1990.

[28] Scozzari, F. Logical optimality of groundness analysis. In *Proceedings of International Static Analysis Symposium, SAS'97* (1997), P. V. Hentenryck, Ed., vol. 1302 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 83–97.

[29] Selke, R. P. A rewriting semantics for program dependence graphs. In *Conference Record of the 16th ACM Symp. on Principles of Programming Languages (*POPL '89*)* (1989), ACM Press, New York, pp. 12–24.

[30] Shmuely, Z. The structure of Galois connections. *Pacific J. Math. 54*, 2 (1974), 209–225.

[31] Smyth, M. B. Topology. In *Handbook of logic in computer science*, vol. 1. Clarendon press - Oxford, 1992.

[32] Weiser, M. Program slicing. *IEEE Transactions on software engineering SE-10*, 4 (1984), 352–357.

[33] Winskel, G. *The formal semantics of programming languages: an introduction*. MIT press, 1993.