Compositional and Inductive Semantic Definitions in Fixpoint, Equational, Constraint, Closure-condition, Rule-based and Game-Theoretic Form *

(invited paper)

Patrick Cousot[1] and Radhia Cousot[2]

cousot@dmi.ens.fr rcousot@lix.polytechnique.fr

[1] LIENS, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris cedex 05, France

[2] LIX, CNRS & École Polytechnique, 91140 Palaiseau cedex, France

Abstract. We present a language and semantics-independent, compositional and inductive method for specifying formal semantics or semantic properties of programs in equivalent fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form. The definitional method is obtained by extending set-theoretic definitions in the context of partial orders. It is parameterized by the language syntax, by the semantic domains and by the semantic transformers corresponding to atomic and compound program components. The definitional method is shown to be preserved by abstract interpretation in either fixpoint, equational, constraint, closure-condition, rule-based or game-theoretic form. The features common to all possible instantiations are factored out thus allowing for results of general scope such as well-definedness, semantic equivalence, soundness and relative completeness of abstract interpretations, etc. to be proved compositionally in a general language and semantics-independent framework.

1. Introduction

Program semantics as well as program proof and analysis methods can be presented in many different styles:

- Fixpoint definitions have been introduced to define the denotational semantics of programming languages (see e.g. [14]);
- Equational definitions are of common use e.g. in context-free grammars [3], abstract interpretation [6], etc;
- Constraint-based definitions are used e.g. in set-based program analysis [11] or in type inference [15];
- A typical use of closure-condition-based definitions is to define sets of terms (see e.g. [19], p. 681);
- Rule-based definitions are used e.g. in Hoare-logic [12], in structured operational semantics [17], type inference [10], program analysis [16];
- Game-theoretic definitions have been used to prove full abstraction for PCF [1].

* This work was partly supported by Esprit BRA 8130 LOMAPS.

We would like to compare these methods for defining program semantics, proofs and program analyses compositionally, by induction on the program syntax both in a language-independent way and in an order-theoretic setting (rather than in the context of set-theory as in [2]).

As far as the language-independent modeling of the semantics of programming languages is concerned, transition systems are a simple model of the operational semantics. No equivalent exists for notions such that Hoare logic or Scott-Strachey denotational semantics. In order to proceed compositionally by induction on the program syntax, these notions are often introduced using a simple programming language [4, 14]. Reasonings using this particular example are not general enough. It follows that, with the notable exception of operational semantics modeled by a (labeled) transition system, formal semantics and program analyses are difficult to present in the abstract, independently of a particular programming language. In this paper we propose a method to cope with such problems.

We show that using a meta-syntax scheme and a meta-semantics scheme, it is possible to propose a general framework for defining the semantics, proofs and analysis of programs compositionally by induction on the meta-syntax. We show that the fixpoint, equational, constraint, closure-condition, rule-based, and game-theoretic styles of definition of the meta-semantics are not fundamentally different but a simple matter of presentation with equivalent interpretations. We next show that this definitional method is preserved by abstract interpretation. This means that the abstraction of a semantics can be presented in the same style as the semantics. Finally this definitional method is shown to be useful for proving general language-independent results such well-definedness, semantic equivalence, soundness and relative completeness of abstract interpretations.

2. Introductory example

Let us first illustrate the definition of the semantics $\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]$ of the μ-expression $\mu X \cdot 0 + aX$ [13] in equivalent fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form. For the sake of conciseness the behaviors of μ-expressions are described by sets of finite sequences of actions.

In fixpoint form, the semantics $\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]$ is the subset of the set $\{0, a\}^\star$ of finite strings on the alphabet $\{0, a\}$ defined as:

$$\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \lambda \mathcal{X} \cdot \{0\} \cup \{a\sigma : \sigma \in \mathcal{X}\}$$

In equivalent equational form, this is the $\subseteq$-least solution to the equation:

$$\mathcal{X} = \{0\} \cup \{a\sigma : \sigma \in \mathcal{X}\}$$

In equivalent set-constraint form, this is the $\subseteq$-least solution to the constraint:

$$\begin{cases} \mathcal{X} \supseteq \{0\} \\ \mathcal{X} \supseteq \{a\sigma : \sigma \in \mathcal{X}\} \end{cases}$$

This can also be written as the $\subseteq$-least $\mathcal{X}$ satisfying the closure-condition:

$$\begin{cases} 0 \in \mathcal{X} \\ \sigma \in \mathcal{X} \;\Rightarrow\; a\sigma \in \mathcal{X} \end{cases}$$

The semantics $\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]$ can also be inductively defined by a formal system with the following axiom and rule schema:

$$0 \in \mathcal{S}\,[\![\mu X \cdot 0 + aX]\!] \qquad \frac{\sigma \in \mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]}{a\sigma \in \mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]}$$

which stands for the formal system with rule instances $\dfrac{\emptyset}{0}$ and $\dfrac{\{\sigma\}}{a\sigma}$, $\sigma \in \{0, a\}^\star$ where the rule instance $\dfrac{P}{c}$ means that from the set P of premises one can infer the conclusion c.
Finally, the semantics $\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]$ can also be inductively defined by the game with rules (presented in tabular and set of pairs forms):

I	II
0	$\emptyset$
$a\sigma$	$\{\sigma\}$

$$\{\langle 0,\ \emptyset\rangle,\ \langle a\sigma,\ \{\sigma\}\rangle \mid \sigma \in \{0, a\}^\star\}$$

The game starts with player I choosing $\sigma = \sigma_1 \in \{0, a\}^\star$. If after n moves player I chooses $\sigma_n \in \{0, a\}^\star$ then player II must choose some X_n such that $\langle \sigma_n,\ X_n\rangle$ is allowed by the rules. The answer of player I must be some $\sigma_{n+1} \in X_n$. A player who is blocked has lost. If the game goes on for ever, player II has lost. The semantics $\mathcal{S}\,[\![\mu X \cdot 0 + aX]\!]$ is the set of initial σ for which player II has a winning strategy in the game.

Such set-theoretic definitions are now extended in the context of partial-orders to proceed compositionally, by induction on the syntax of programs.

3. Syntax Scheme

Following [7], we let $\mathcal{L}$ be a non-empty set of program components (or fragments) and $\mathcal{P} \subseteq \mathcal{L}$ be the non-empty subset of complete programs. We let $\Gamma\,[\![\cdot]\!] \in \mathcal{L} \mapsto \wp(\mathcal{L})$ define the set of immediate strict components of a program fragment. This set must be finite:

$$\forall \pi \in \mathcal{L} : \Gamma\,[\![\pi]\!] \text{ is finite.}$$

No program fragment can be indefinitely decomposed into strictly smaller components:

> There is no infinite chain $\pi_0, \pi_1, \ldots, \pi_i, \ldots$ in $\mathcal{L}$ such that $\forall i \geq 0 : \pi_{i+1} \in \Gamma\,[\![\pi_i]\!]$.

Subcomponents of a program fragment are all different (this can be obtained e.g. by labeling):

> If there are chains $\pi_0, \ldots, \pi_m$ and $\pi'_0, \ldots, \pi'_n$ in $\mathcal{L}$ such that for all $0 \leq i < m : \pi_{i+1} \in \Gamma\,[\![\pi_i]\!]$, for all $0 \leq j < n : \pi'_{j+1} \in \Gamma\,[\![\pi'_j]\!]$, $\pi_0 = \pi'_0$ and $\pi_m = \pi'_n$ then $m = n$ and $\forall i \in [0, n] : \pi_i = \pi'_i$.

The set of all subcomponents $\Gamma^\star\,[\![\cdot]\!] \in \mathcal{L} \mapsto (\wp(\mathcal{L}) - \{\emptyset\})$ of a program fragment is:

$$\Gamma^\star\,[\![\pi]\!] \stackrel{\text{def}}{=} \{\pi' : \exists n \geq 0 : \exists \pi_0, \ldots, \pi_n \in \mathcal{L} : (\pi = \pi_0) \wedge (\forall i \in [0, n[: \pi_{i+1} \in \Gamma\,[\![\pi_i]\!]) \wedge (\pi_n = \pi')\}$$

The set of all atomic subcomponents $\Gamma_a^\star [\![\cdot]\!] \in \mathcal{L} \longmapsto (\wp(\mathcal{L}) - \{\emptyset\})$ of a program fragment is:

$$\Gamma_a^\star [\![\pi]\!] \stackrel{\text{def}}{=} \{\pi' \in \Gamma^\star [\![\pi]\!] : \Gamma [\![\pi']\!] = \emptyset\}$$

The set of all compound subcomponents $\Gamma_c^\star [\![\cdot]\!] \in \mathcal{L} \longmapsto (\wp(\mathcal{L}))$ of a program fragment is:

$$\Gamma_c^\star [\![\pi]\!] \stackrel{\text{def}}{=} \{\pi' \in \Gamma^\star [\![\pi]\!] : \Gamma [\![\pi']\!] \neq \emptyset\}$$

Example The immediate components of the μ-expression $\mu X \cdot 0 + aX$ are $\Gamma [\![\mu X \cdot 0 + aX]\!] = \{0{+}aX\}$. Its subcomponents are $\Gamma^\star [\![\mu X \cdot 0 + aX]\!] = \{\mu X \cdot 0{+}aX, 0{+}aX, 0, aX, a, X\}$. Its atomic components are $\Gamma_a^\star [\![\mu X \cdot 0 + aX]\!] = \{0, a, X\}$. Its compound components are $\Gamma_c^\star [\![\mu X \cdot 0 + aX]\!] = \{\mu X \cdot 0 + aX, 0 + aX, aX\}$. Depending upon the structural induction which is used other decompositions may also be used. □

The following proposition is useful to justify definitions and proofs by structural induction on the syntax of programs:

Proposition 1. *For all programs $\Pi \in \mathcal{P}$, the binary relation $\pi' \prec \pi \stackrel{\text{def}}{=} \pi' \in \Gamma [\![\pi]\!]$ and $\pi' \preceq \pi \stackrel{\text{def}}{=} (\pi' \prec \pi) \vee (\pi' = \pi)$ is a well-founded partial ordering on $\Gamma^\star [\![\Pi]\!]$.*

4. Semantic Domains Scheme

The definition of the semantic scheme of program $\Pi \in \mathcal{P}$ is parameterized by a semantic domain $\mathcal{D}_\pi$ associated with each component $\pi \in \Gamma^\star [\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$, such that:

Hypothesis 2. *The semantic domain $\mathcal{D}_\pi$, $\pi \in \Gamma^\star [\![\Pi]\!]$ is a complete partial order (cpo) $(\mathcal{D}_\pi, \sqsubseteq, \bot, \bigsqcup)$ so that $\sqsubseteq$-increasing chains have a least upper-bound (lub) denoted $\bigsqcup$.*

5. Semantic Transformers Scheme

The definition of the semantic scheme of program Π is parameterized by monotonic semantic transformers $\mathcal{F}_\pi$ associated with each subcomponent π of Π. Their signatures are defined by induction on the syntax of the program Π:

Hypothesis 3. *The semantic transformers $\mathcal{F}_\pi$ of all components $\pi \in \Gamma^\star [\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ satisfy:*

– *for atomic program components $\pi \in \Gamma_a^\star [\![\Pi]\!]$, $\mathcal{F}_\pi \in \mathcal{D}_\pi \xrightarrow{\text{m}} \mathcal{D}_\pi$*

– *for compound program components $\pi \in \Gamma_c^\star [\![\Pi]\!]$:*

$$\mathcal{F}_\pi \in \left(\prod\nolimits_{\pi' \in \Gamma [\![\pi]\!]} \mathcal{D}_{\pi'}\right) \xrightarrow{\text{m}} (\mathcal{D}_\pi \xrightarrow{\text{m}} \mathcal{D}_\pi)$$

6. Semantic Scheme

The semantic scheme of a program Π associates an element of the semantic domain $\mathcal{D}_\pi$ to each component π of Π. We consider several styles of presentation of this semantics by structural induction on the syntax of the program and prove them to be equivalent.

6.1 Fixpoints

The fixpoint definition of the semantics uses the least fixpoint operator lfp $\in (L \overset{\text{m}}{\longmapsto} L) \overset{\text{m}}{\longmapsto} L$ such that given $F \in L \overset{\text{m}}{\longmapsto} L$ on the poset $(L, \sqsubseteq)$:

$$F(\operatorname{lfp} F) = \operatorname{lfp} F \qquad \text{if } F(X) = X \text{ then } \operatorname{lfp} F \sqsubseteq X \tag{1}$$

For all monotonic operators $F \in L \overset{\text{m}}{\longmapsto} L$ on a cpo $(L, \sqsubseteq, \bot, \bigsqcup)$, lfp exists and is such that ($\mathbb{O}$ is the class of ordinals) $\operatorname{lfp} F = \bigsqcup_{\kappa \in \mathbb{O}} F^\kappa$ where the *approximants* are $F^\kappa \overset{\text{def}}{=} F\Big(\bigsqcup_{\lambda < \kappa} F^\lambda\Big)$ where $\sqcup\emptyset = \bot$. Let us also recall the fixpoint induction proof method. For all $X \in L$:

$$F(X) \sqsubseteq X \Rightarrow \operatorname{lfp} F \sqsubseteq X \tag{2}$$

6.2 Fixpoint semantic definition scheme

The fixpoint semantic scheme is parameterized by semantic domains $\mathcal{D}_\pi$ and semantic transformers $\mathcal{F}_\pi$ associated with each program subcomponent $\pi \in \Gamma^\star [\![\Pi]\!]$ according to Hyp. 2 and Hyp. 3. It is defined compositionally, by induction on the syntax of the program Π.

Definition 4. The fixpoint semantics $\mathcal{S}_{\text{fi}} [\![\pi]\!]$, $\pi \in \Gamma^\star [\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ is defined such that:

- for atomic program components $\pi \in \Gamma^\star_a [\![\Pi]\!]$, $\mathcal{S}_{\text{fi}} [\![\pi]\!] \overset{\text{def}}{=} \operatorname{lfp} \mathcal{F}_\pi$;
- for compound program components $\pi \in \Gamma^\star_c [\![\Pi]\!]$:

$$\mathcal{S}_{\text{fi}} [\![\pi]\!] \overset{\text{def}}{=} \operatorname{lfp} \mathcal{F}_\pi \Big(\prod_{\pi' \in \Gamma [\![\pi]\!]} \mathcal{S}_{\text{fi}} [\![\pi']\!]\Big)$$

The fixpoint semantics $\mathcal{S}_{\text{fi}} [\![\Pi]\!]$ is well-defined:

Proposition 5. *For all $\pi \in \Gamma^\star [\![\Pi]\!]$, $\mathcal{S}_{\text{fi}} [\![\pi]\!] \in \mathcal{D}_\pi$.*

6.3 Equational semantic definition scheme

The definition of the equational semantics $\mathcal{S}_{\text{eq}} [\![\pi]\!]$, $\pi \in \Gamma^\star [\![\Pi]\!]$ uses variables $\mathcal{X}_\pi$ associated with each component π of program Π chosen such that $\pi \neq \pi' \iff \mathcal{X}_\pi \neq \mathcal{X}_{\pi'}$. This definition is compositional, by induction on the syntax of the program Π.

Definition 6. The equational semantics $\mathcal{S}_{\text{eq}} [\![\pi]\!]$, $\pi \in \Gamma^\star [\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ is the $\sqsubseteq$-least solution to the following system of equations:

- the semantic equation corresponding to an atomic program component $\pi \in \Gamma^\star_a [\![\Pi]\!]$ is $\mathcal{X}_\pi = \mathcal{F}_\pi(\mathcal{X}_\pi)$;
- the semantic equation corresponding to a compound program component $\pi \in \Gamma^\star_c [\![\Pi]\!]$ is:

$$\mathcal{X}_\pi = \mathcal{F}_\pi \Big(\prod_{\pi' \in \Gamma [\![\pi]\!]} \mathcal{X}_{\pi'}\Big)(\mathcal{X}_\pi)$$

Proposition 7. *For all $\pi \in \Gamma^\star [\![\Pi]\!]$, the equational definition of the semantics of program component π is equivalent to its fixpoint definition: $\mathcal{S}_{\text{eq}} [\![\pi]\!] = \mathcal{S}_{\text{fi}} [\![\pi]\!]$.*

6.4 Constraint-based semantic definition scheme

By the fixpoint induction proof method (2), lfp F is the $\sqsubseteq$-least solution to the constraint $X \sqsupseteq F(X)$. This remark leads to the definition of the constraint-based semantics $\mathcal{S}_{\text{co}}[\![\pi]\!]$, $\pi \in \Gamma^{\star}[\![\Pi]\!]$. Again the definition is compositional, by induction on the syntax of the program:

Definition 8. The constraint-based semantics $\mathcal{S}_{\text{co}}[\![\pi]\!]$, $\pi \in \Gamma^{\star}[\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ is the $\sqsubseteq$-least solution to the following system of semantic constraints:
- the semantic constraint corresponding to an atomic program component $\pi \in \Gamma_a^{\star}[\![\Pi]\!]$ is $\mathcal{X}_\pi \sqsupseteq \mathcal{F}_\pi(\mathcal{X}_\pi)$;
- the semantic constraint corresponding to a compound program component $\pi \in \Gamma_c^{\star}[\![\Pi]\!]$ is $\mathcal{X}_\pi \sqsupseteq \mathcal{F}_\pi\Big(\prod_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{X}_{\pi'}\Big)(\mathcal{X}_\pi)$.

Usually the constraint can be decomposed into a system of more elementary set-constraints using simple set-theoretic algebraic identities. For example:

$$X \bigsqcup Y \sqsubseteq Z \iff X \sqsubseteq Z \wedge Y \sqsubseteq Z$$

and, in a complete lattice, if $F(X) = \bigsqcup\{g(x) : f(x) \sqsubseteq X\}$ then:

$$X \sqsupseteq F(X) \iff \forall x : f(x) \sqsubseteq X \Rightarrow g(x) \sqsubseteq X$$

Proposition 9. *For all program components $\pi \in \Gamma^{\star}[\![\Pi]\!]$, the constraint-based definition of the semantics of π is equivalent to its fixpoint definition: $\mathcal{S}_{\text{co}}[\![\pi]\!] = \mathcal{S}_{\text{fi}}[\![\pi]\!]$.*

6.5 Closure semantic definition scheme

Given a poset $(L, \sqsubseteq)$, a *closure-condition* is $C \in \wp(L \times L)$ which is monotonic in its second component, that is, for all $x, X, Y \in L$, $C(x, X) \wedge X \sqsubseteq Y \Rightarrow C(x, Y)$ where $C(x, X)$ is true if and only if $\langle x,\ X\rangle \in C$. A *closure-definition* has the form:

$\boldsymbol{X}$ is the $\sqsubseteq$-least element X of L satisfying: $\forall x \in L : C(x, X) \Rightarrow x \sqsubseteq X$

The closure-definition is said to be *well-formed* if $\boldsymbol{X}$ exists. This order-theoretic definition generalizes the usual set-theoretic definition [2] of the least set X of L such that $\forall x \in L : C(x, X) \Rightarrow x \in X$.

A closure-definition can be presented in fixpoint form:

Proposition 10. *If $(L, \sqsubseteq)$ is a cpo and $\Xi(X) \stackrel{\text{def}}{=} \bigsqcup\{x \in L : C(x, X)\}$ is well-defined then the closure-definition is well-formed and $\boldsymbol{X} = \mathit{lfp}\, \Xi$.*

Reciprocally, a fixpoint definition can be presented as a closure-definition:

Proposition 11. *If $L(\sqsubseteq, \bot, \bigsqcup)$ is a cpo and $F \in L \overset{\text{m}}{\longmapsto} L$ then the closure-definition with condition $C(x, X) = x \sqsubseteq F(X)$ is well-formed and defines $\mathit{lfp}\, F$.*

This leads to the compositional definition of the closure-condition-based semantics $\mathcal{S}_{\text{cl}}[\![\pi]\!]$, $\pi \in \Gamma^{\star}[\![\Pi]\!]$, by induction on the syntax of the program:

Definition 12. The closure-condition-based semantics $\mathcal{S}_{\text{cl}}[\![\pi]\!]$, $\pi \in \Gamma^{\star}[\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ is the $\sqsubseteq$-least element $\mathcal{X}_\pi$ of $\mathcal{D}_\pi$ satisfying the following closure-condition $C_\pi(x, \mathcal{X}_\pi)$:

- the closure-condition $C_\pi(x, \mathcal{X}_\pi)$ corresponding to an atomic program component $\pi \in \Gamma_a^\star [\![\Pi]\!]$ is $x \sqsubseteq \mathcal{F}_\pi(\mathcal{X}_\pi)$;
- the closure-condition $C_\pi(x, \mathcal{X}_\pi)$ corresponding to a compound program component $\pi \in \Gamma_c^\star [\![\Pi]\!]$ is $x \sqsubseteq \mathcal{F}_\pi\Big(\prod_{\pi' \in \Gamma [\![\pi]\!]} \mathcal{X}_{\pi'}\Big)(\mathcal{X}_\pi)$.

Proposition 13. *For all program components $\pi \in \Gamma^\star [\![\Pi]\!]$, the closure-condition-based definition of the semantics of π is equivalent to its fixpoint definition: $\mathcal{S}_{\text{cl}} [\![\pi]\!] = \mathcal{S}_{\text{fi}} [\![\pi]\!]$.*

6.6 Rule-based formal systems

The semantics can also be specified by a formal system based on a poset $\langle L, \sqsubseteq \rangle$ with rule instances :

$$R = \left\{ \frac{P_i}{C_i} : i \in \Delta \right\}$$

such that for all $i \in \Delta$, $P_i \in L$ and $C_i \in L$ [8]. By definition, this denotes:

$$\text{lfp}\, \Phi \tag{3}$$

where the *R-operator* Φ is $\Phi \stackrel{\text{def}}{=} \lambda X \cdot \bigsqcup \{C_i : \exists i \in \Delta : P_i \sqsubseteq X\}$. R is well-defined if and only if:

$$\forall X \in L : \bigsqcup \{C_i : \exists i \in \Delta : P_i \sqsubseteq X\} \text{ exists} \tag{4}$$

which is the case e.g. if $\langle L, \sqsubseteq \rangle$ is a complete lattice.

Proposition 14. (4) *implies that Φ is monotonic hence that* (3) *is well-defined.*

This generalizes the set-theoretic formal systems considered in [2] where L is $(\wp(U), \subseteq, \emptyset, U, \cup, \cap)$ for a given universe U. Rules in [2] are written:

$$\left\{ \frac{P_i}{c_i} : i \in \Delta \right\}$$

where $P_i \subseteq U$ and $c_i \in U$. Their meaning is defined to be $\text{lfp}\, \Psi$ where $\Psi \stackrel{\text{def}}{=} \lambda X \cdot \{c_i : \exists i \in \Delta : P_i \subseteq X\}$. In an order-theoretic setting, we would write them:

$$\left\{ \frac{P_i}{\{c_i\}} : i \in \Delta \right\}$$

with equivalent meaning $\text{lfp}\, \Phi$ since: $\Phi = \lambda X \cdot \bigcup \{\{c_i\} : \exists i \in \Delta : P_i \subseteq X\} = \Psi$.

6.7 Rule-based semantic definition scheme

Again, the rule-based semantics $\mathcal{S}_{\text{ru}} [\![\Pi]\!]$ is defined compositionally, by induction on the syntax of the program Π. In practice the formal system uses axioms (with $P = \bot$) and rule schemata which are interpreted as rule instances.

Definition 15. The rule-based semantics $\mathcal{S}_{\text{ru}} [\![\pi]\!]$, $\pi \in \Gamma^\star [\![\Pi]\!]$ is defined by the following rule instances:

– for atomic program components $\pi \in \Gamma_a^\star [\![\Pi]\!]$:

$$\begin{cases} \dfrac{P}{C}\,\pi \\ P \subseteq \mathcal{D}_\pi \wedge C \sqsubseteq \mathcal{F}_\pi(P) \end{cases}$$

– for compound program components $\pi \in \Gamma_c^\star [\![\Pi]\!]$:

$$\begin{cases} \dfrac{P}{C}\,\pi \\ P \subseteq \mathcal{D}_\pi \wedge C \sqsubseteq \mathcal{F}_\pi\Big(\prod_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{S}_{\mathrm{ru}}[\![\pi']\!]\Big)(P) \end{cases}$$

Proposition 16. *For all $\pi \in \Gamma^\star [\![\Pi]\!]$, the rule-based definition of the semantics of program component π is equivalent to its fixpoint definition: $\mathcal{S}_{\mathrm{ru}}[\![\pi]\!] = \mathcal{S}_{\mathrm{fi}}[\![\pi]\!]$.*

6.8 Games

Given a poset $\langle L, \sqsubseteq\rangle$, a game is defined by rules $R \subseteq L \times L$. The rules are well-defined if and only if $\forall X \in L : \bigsqcup\{C : \exists\langle C, P\rangle \in R : P \sqsubseteq X\}$ exists. The corresponding *R-operator* Φ is $\Phi \stackrel{\mathrm{def}}{=} \lambda X \cdot \bigsqcup\{C : \exists\langle C, P\rangle \in R : P \sqsubseteq X\}$. The game $\mathcal{G}(R, a)$ with rules R starting from initial position $a \in L$ is played by two players I and II. Player I must start by choosing $x_0 = a$. If player I chooses x_n in the n-th move, then player II must respond by $X_n \in \wp(L)$ such that $x_n = \Phi(\bigsqcup X_n)$. For the next move, player I must choose some $x_{n+1} \in X_n$. A player who is blocked has lost. If the game goes on forever then player II has lost. We define $\mathcal{W}(R)$ as the set of initial winning positions for player II:

$$\mathcal{W}(R) \stackrel{\mathrm{def}}{=} \{a \in L : \text{player II has a winning strategy in game } \mathcal{G}(R, a)\}$$

Proposition 17. *If the rules R are well-defined then $\mathit{lfp}\,\Phi = \bigsqcup \mathcal{W}(R)$*

Now a fixpoint definition can be given an equivalent game-theoretic form:

Proposition 18. *If $\langle L, \sqsubseteq\rangle$ is a cpo and $F \in L \stackrel{\mathrm{m}}{\longmapsto} L$ is monotonic then $\mathit{lfp}\,F = \bigsqcup \mathcal{W}(R)$ for the game with rules $R = \{\langle C, P\rangle : P \in L \wedge C \sqsubseteq F(P)\}$.*

6.9 Game-theoretic semantic definition scheme

Again, the game-theoretic semantics $\mathcal{S}_{\mathrm{ga}}[\![\Pi]\!]$ is defined compositionally, by induction on the syntax of the program Π.

Definition 19. The game-theoretic semantics $\mathcal{S}_{\mathrm{ga}}[\![\pi]\!]$, $\pi \in \Gamma^\star [\![\Pi]\!]$ is defined by the following rules R_π:

– for atomic program components $\pi \in \Gamma_a^\star [\![\Pi]\!]$:

$$R_\pi = \{\langle C, P\rangle : P \sqsubseteq \mathcal{D}_\pi \wedge C \sqsubseteq \mathcal{F}_\pi(P)\}$$

– for compound program components $\pi \in \Gamma_c^\star [\![\Pi]\!]$:

$$R_\pi = \Big\{\langle C, P\rangle : P \sqsubseteq \mathcal{D}_\pi \wedge C \sqsubseteq \mathcal{F}_\pi\Big(\prod_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{S}_{\mathrm{ga}}[\![\pi']\!]\Big)(P)\Big\}$$

Proposition 20. *For all $\pi \in \Gamma^\star [\![\Pi]\!]$, the game-theoretic definition of the semantics of program component π is equivalent to its fixpoint definition: $\mathcal{S}_{\mathrm{ga}}[\![\pi]\!] = \mathcal{S}_{\mathrm{fi}}[\![\pi]\!]$.*

6.10 Equivalence of the semantic definitions

We conclude that the fixpoint, equational, constraint-based, closure-condition-based, rule-based and game-theoretic semantic definitions are all equivalent:

Proposition 21. *For all components* $\pi \in \Gamma^\star [\![\Pi]\!]$ *of program* $\Pi \in \mathcal{P}$, $\mathcal{S}_{\text{fi}} [\![\pi]\!] = \mathcal{S}_{\text{eq}} [\![\pi]\!] = \mathcal{S}_{\text{co}} [\![\pi]\!] = \mathcal{S}_{\text{cl}} [\![\pi]\!] = \mathcal{S}_{\text{ru}} [\![\pi]\!]$.

7. Example

μ-expressions [13] provide a simple example of semantics definition. μ-expressions are defined by the following grammar:

$$E ::= 0 \mid X \mid aE \mid E_1 + E_2 \mid \mu X \cdot E$$

where $a \in \mathsf{A}$ is an action and $X \in \mathsf{V}$ is a variable.

The immediate strict components $\Gamma [\![E]\!]$ of μ-expression E are:

$$\begin{array}{ccc} \Gamma [\![0]\!] = \emptyset & \Gamma [\![aE]\!] = \{E\} & \Gamma [\![X]\!] = \emptyset \\ \Gamma [\![E_1 + E_2]\!] = \{E_1, E_2\} & \Gamma [\![a]\!] = \emptyset & \Gamma [\![\mu X \cdot E]\!] = \{E\} \end{array}$$

We define the following semantic domains:

$a : \mathbb{A} \stackrel{\text{def}}{=} \{0\} \cup \mathsf{A}$	action alphabet
$\sigma : \mathbb{S} \stackrel{\text{def}}{=} \mathbb{A}^+$	nonempty finite strings
$v : \mathbb{V} \stackrel{\text{def}}{=} \wp(\mathbb{S})$	values
$\rho : \mathbb{E} \stackrel{\text{def}}{=} \mathsf{V} \longmapsto \mathbb{V}$	environments
$\varphi, \psi : \mathbb{D} \stackrel{\text{def}}{=} \mathbb{E} \longmapsto \mathbb{V}$	semantic domain
$\phi : \mathbb{T}_a \stackrel{\text{def}}{=} \mathbb{D} \xrightarrow{\text{m}} \mathbb{D}$	atomic semantic transformer domain
$\Phi : \mathbb{T}_c{}^n \stackrel{\text{def}}{=} \mathbb{D}^n \xrightarrow{\text{m}} \mathbb{T}_a$	$n > 0$, compound semantical transformer domain

The semantic domain $(\mathbb{D}, \dot\subseteq, \lambda\rho \cdot \emptyset, \lambda\rho \cdot \mathbb{S}, \dot\cup, \dot\cap)$ is a complete lattice hence a cpo for the pointwise partial ordering $\dot\subseteq$.

In the definition of the fixpoint semantics $\mathcal{S}_{\text{fi}} [\![E]\!] \in \mathbb{D}$ of μ-expression E, we have $\rho[X := v](X) = v$ while $\rho[X := v](Y) = \rho(Y)$ when $X \neq Y$:

$$\begin{array}{ll} \mathcal{S}_{\text{fi}} [\![0]\!] \stackrel{\text{def}}{=} \lambda\rho \cdot \{0\} & \mathcal{S}_{\text{fi}} [\![E_1 + E_2]\!] \stackrel{\text{def}}{=} \mathcal{S}_{\text{fi}} [\![E_1]\!] \,\dot\cup\, \mathcal{S}_{\text{fi}} [\![E_2]\!] \\ \mathcal{S}_{\text{fi}} [\![X]\!] \stackrel{\text{def}}{=} \lambda\rho \cdot \rho(X) & \mathcal{S}_{\text{fi}} [\![\mu X \cdot E]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \lambda\mathcal{X} \cdot \lambda\rho \cdot \mathcal{S}_{\text{fi}} [\![E]\!] \rho[X := \mathcal{X}(\rho)] \\ \mathcal{S}_{\text{fi}} [\![aE]\!] \stackrel{\text{def}}{=} \lambda\rho \cdot \{a\sigma : \sigma \in \mathcal{S}_{\text{fi}} [\![E]\!] \rho\} & \end{array}$$

These fixpoint definitions can be written in the form required by Hyp. 3:

$$\begin{array}{ll} \mathcal{S}_{\text{fi}} [\![0]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F}_0 & \mathcal{S}_{\text{fi}} [\![E_1 + E_2]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F}_{E_1+E_2}(\langle \mathcal{S}_{\text{fi}} [\![E_1]\!], \mathcal{S}_{\text{fi}} [\![E_2]\!] \rangle) \\ \mathcal{S}_{\text{fi}} [\![X]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F}_X & \mathcal{S}_{\text{fi}} [\![\mu X \cdot E]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F}_{\mu X \cdot E}(\mathcal{S}_{\text{fi}} [\![E]\!]) \\ \mathcal{S}_{\text{fi}} [\![aE]\!] \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F}_{aE}(\mathcal{S}_{\text{fi}} [\![E]\!]) & \end{array}$$

by defining the following semantic transformers:

$$\begin{array}{ll} \mathcal{F}_0 \stackrel{\text{def}}{=} \lambda\mathcal{X} \cdot \lambda\rho \cdot \{0\} & \mathcal{F}_{E_1+E_2} \stackrel{\text{def}}{=} \lambda\langle \varphi, \psi \rangle \cdot \lambda\mathcal{X} \cdot \varphi \,\dot\cup\, \psi \\ \mathcal{F}_X \stackrel{\text{def}}{=} \lambda\mathcal{X} \cdot \lambda\rho \cdot \rho(X) & \mathcal{F}_{\mu X \cdot E} \stackrel{\text{def}}{=} \lambda\langle \varphi \rangle \cdot \lambda\mathcal{X} \cdot \lambda\rho \cdot \varphi(\rho[X := \mathcal{X}(\rho)]) \\ \mathcal{F}_{aE} \stackrel{\text{def}}{=} \lambda\langle \varphi \rangle \cdot \lambda\mathcal{X} \cdot \lambda\rho \cdot \{a\sigma : \sigma \in \varphi(\rho)\} & \end{array} \tag{5}$$

In the equational definition of the semantics, we use the usual convention of naming $[\![E]\!]$ the variable associated with program component E:

$$\begin{array}{ll} [\![0]\!] = \lambda\rho\cdot\{0\} & [\![E_1 + E_2]\!] = [\![E_1]\!] \dot{\cup} [\![E_2]\!] \\ [\![X]\!] = \lambda\rho\cdot \rho(X) & [\![\mu X\cdot E]\!] = \lambda\rho\cdot [\![E]\!]\rho[X := [\![\mu X\cdot E]\!](\rho)] \\ [\![aE]\!] = \lambda\rho\cdot\{a\sigma : \sigma \in [\![E]\!]\rho\} & \end{array}$$

For the constraint-based definition we use simple identities such as $\lambda\rho\cdot\{x\} \dot{\subseteq} \mathcal{X}$ if and only if $\forall\rho : x \in \mathcal{X}(\rho)$ and free variables are universally quantified:

$$\begin{array}{rr} 0 \in [\![0]\!]\rho & [\![E_1]\!] \dot{\subseteq} [\![E_1 + E_2]\!] \\ \rho(X) \subseteq [\![X]\!]\rho & [\![E_2]\!] \dot{\subseteq} [\![E_1 + E_2]\!] \\ \{a\sigma : \sigma \in [\![E]\!]\rho\} \subseteq [\![aE]\!]\rho & [\![E]\!]\rho[X := [\![\mu X\cdot E]\!](\rho)] \subseteq [\![\mu X\cdot E]\!]\rho \end{array}$$

For the closure-condition-based definition we use the identity $X \subseteq Y$ if and only if $\forall x \in X : x \in Y$. Free variables are universally quantified:

$$\begin{array}{rr} 0 \in [\![0]\!]\rho & \sigma \in [\![E_1]\!] \Rightarrow \sigma \in [\![E_1 + E_2]\!] \\ \sigma \in \rho(X) \Rightarrow \sigma \in [\![X]\!]\rho & \sigma \in [\![E_2]\!] \Rightarrow \sigma \in [\![E_1 + E_2]\!] \\ \sigma \in [\![E]\!]\rho \Rightarrow a\sigma \in [\![aE]\!]\rho & \sigma \in [\![E]\!]\rho[X := [\![\mu X\cdot E]\!](\rho)] \Rightarrow \sigma \in [\![\mu X\cdot E]\!]\rho \end{array}$$

For the rule-based definition of the semantics $\mathcal{S}_{\mathrm{ru}}[\![E]\!]$ of μ-expression E, the axioms and rule schemata of the formal system are:

$$0 \in \mathcal{S}_{\mathrm{ru}}[\![0]\!]\rho \qquad \frac{\sigma \in \rho(X)}{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![X]\!]\rho} \qquad \frac{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E]\!]\rho}{a\sigma \in \mathcal{S}_{\mathrm{ru}}[\![aE]\!]\rho} \qquad \frac{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E_1]\!]\rho}{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E_1 + E_2]\!]\rho}$$

$$\frac{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E_2]\!]\rho}{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E_1 + E_2]\!]\rho} \qquad \frac{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![E]\!]\rho[X := \mathcal{S}_{\mathrm{ru}}[\![\mu X\cdot E]\!](\rho)]}{\sigma \in \mathcal{S}_{\mathrm{ru}}[\![\mu X\cdot E]\!]\rho}$$

The interpretation of these axioms and rule schemata in terms of rule instances, the meaning of which is provided by (3), is as follows:

$$\frac{\emptyset}{\{\lambda\rho\cdot\{0\}\}}{\scriptstyle 0} \qquad \frac{\emptyset}{\{\lambda\rho\cdot \rho(X)\}}{\scriptstyle X} \qquad \frac{\emptyset}{\{\lambda\rho\cdot\{a\sigma : \sigma \in \mathcal{S}_{\mathrm{ru}}[\![E]\!]\rho\}\}}{\scriptstyle aE}$$

$$\frac{\emptyset}{\mathcal{S}_{\mathrm{ru}}[\![E_1]\!]}{\scriptstyle E_1+E_2} \qquad \frac{\emptyset}{\mathcal{S}_{\mathrm{ru}}[\![E_2]\!]}{\scriptstyle E_1+E_2} \qquad \frac{P}{\{\lambda\rho\cdot \sigma\}}{\scriptstyle \mu X\cdot E} \quad \text{for all } P \subseteq \mathbb{S} \text{ and} \sigma \in \mathcal{S}_{\mathrm{ru}}[\![E]\!]\rho[X := P]$$

For the game-theoretic definition of the semantics $\mathcal{S}_{\mathrm{ga}}[\![E]\!]$ of μ-expression E, the rules are:

$$R_0 = \{\langle\{\lambda\rho\cdot\{0\}\},\ \emptyset\rangle\} \qquad R_X = \{\langle\{\lambda\rho\cdot \rho(X)\},\ \emptyset\rangle\}$$

$$R_{aE} = \{\langle\{\lambda\rho\cdot\{a\sigma : \sigma \in \mathcal{S}_{\mathrm{ga}}[\![E]\!]\rho\},\ \emptyset\rangle\} \quad R_{E_1+E_2} = \{\langle\mathcal{S}_{\mathrm{ga}}[\![E_1]\!],\ \emptyset\rangle,\ \langle\mathcal{S}_{\mathrm{ga}}[\![E_2]\!],\ \emptyset\rangle\}$$

$$R_{\mu X\cdot E} = \{\langle\{\lambda\rho\cdot \sigma\},\ P\rangle : P \subseteq \mathbb{S}\ \wedge\ \sigma \in \mathcal{S}_{\mathrm{ga}}[\![E]\!]\rho[X := P]\}$$

8. Abstraction Scheme

We now show that abstract interpretation preserves the fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic definitional method for specifying abstract semantic properties of programs.

8.1 Galois Connections

The abstraction process is formalized using Galois connections between posets [6, 9] .

Definition 22. A Galois connection between posets $(L,\ \sqsubseteq)$ and $(L^\sharp,\ \sqsubseteq^\sharp)$ is a pair $\langle\alpha,\ \gamma\rangle$ of functions $\alpha \in L \mapsto L^\sharp$ and $\gamma \in L^\sharp \mapsto L$ such that for all $x \in L$ and $y \in L^\sharp$ $\alpha(x) \sqsubseteq^\sharp y \iff x \sqsubseteq \gamma(y)$ which is denoted as $(L, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L^\sharp, \sqsubseteq^\sharp)$.

In a Galois connection, α is surjective if and only if γ is injective if and only if $\alpha \circ \gamma = 1$ (where $1 = \lambda x\boldsymbol{\cdot}\, x$).

8.2 Abstract Domains Scheme

Given a semantics $\mathcal{S}_{\text{fi}}[\![\pi]\!] \in \mathcal{D}_\pi$, $\pi \in \Gamma^\star[\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$, we consider an abstract interpretation given by Galois connections [6]:

Hypothesis 23. *The abstract semantic domains* $\mathcal{D}^\sharp_\pi$, $\pi \in \Gamma^\star[\![\Pi]\!]$ *are posets* $(\mathcal{D}^\sharp_\pi, \sqsubseteq^\sharp)$ *such that* $(\mathcal{D}_\pi, \sqsubseteq) \underset{\alpha_\pi}{\overset{\gamma_\pi}{\leftrightarrows}} (\mathcal{D}^\sharp_\pi, \sqsubseteq^\sharp)$.

In a Galois connection, α preserves lubs so that if $(L, \sqsubseteq)$ is a cpo and $(L, \sqsubseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (L^\sharp, \sqsubseteq^\sharp)$ then $(\alpha(L), \sqsubseteq^\sharp)$ is also a cpo . It follows that by restricting $\mathcal{D}^\sharp_\pi$ to $\alpha_\pi(\mathcal{D}_\pi)$, Hyp. 23 implies that all abstract semantic domains are cpos:

Proposition 24. *For all* $\pi \in \Gamma^\star[\![\Pi]\!]$, *if* $\mathcal{D}^\sharp_\pi \stackrel{\text{def}}{=} \alpha_\pi(\mathcal{D}_\pi)$ *(where* $\varphi(L) = \{\varphi(x) : x \in L\}$*) then* $(\mathcal{D}^\sharp_\pi, \sqsubseteq^\sharp, \bot^\sharp, \sqcup^\sharp)$ *is a cpo with* $\bot^\sharp \stackrel{\text{def}}{=} \alpha_\pi(\bot)$ *and* $\sqcup^\sharp X \stackrel{\text{def}}{=} \alpha_\pi(\bigsqcup_{x \in X} \gamma_\pi(x))$.

8.3 Abstract Transformers Scheme

The abstraction $\langle\alpha_\pi,\ \gamma_\pi\rangle$ can be lifted to higher-order monotonic functionals:

$$(\mathcal{D}_\pi \xmapsto{\text{m}} \mathcal{D}_\pi, \vec{\sqsubseteq}) \underset{\vec{\alpha}_\pi}{\overset{\vec{\gamma}_\pi}{\leftrightarrows}} (\mathcal{D}^\sharp_\pi \xmapsto{\text{m}} \mathcal{D}^\sharp_\pi, \vec{\sqsubseteq}^\sharp)$$

by defining the functional abstraction [5]:

$$\vec{\alpha}_\pi \stackrel{\text{def}}{=} \lambda F\boldsymbol{\cdot}\, \alpha_\pi \circ F \circ \gamma_\pi \qquad \vec{\gamma}_\pi \stackrel{\text{def}}{=} \lambda F^\sharp\boldsymbol{\cdot}\, \gamma_\pi \circ F^\sharp \circ \alpha_\pi$$

The same way for products:

$$\left(\left(\prod\nolimits_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{D}_{\pi'}\right), \dot{\sqsubseteq}\right) \underset{\dot{\alpha}_\pi}{\overset{\dot{\gamma}_\pi}{\leftrightarrows}} \left(\left(\prod\nolimits_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{D}^\sharp_{\pi'}\right), \dot{\sqsubseteq}^\sharp\right)$$

we define the product abstraction [6]:

$$\dot{\alpha}_\pi \stackrel{\text{def}}{=} \lambda \mathcal{X}\boldsymbol{\cdot} \prod\nolimits_{\pi' \in \Gamma[\![\pi]\!]} \alpha_{\pi'}(\mathcal{X}_{\pi'}) \qquad \dot{\gamma}_\pi \stackrel{\text{def}}{=} \lambda \mathcal{X}^\sharp\boldsymbol{\cdot} \prod\nolimits_{\pi' \in \Gamma[\![\pi]\!]} \gamma_{\pi'}(\mathcal{X}^\sharp_{\pi'})$$

Combining the product and functional abstractions:

$$\left(\left(\prod_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{D}_{\pi'}\right) \xmapsto{\text{m}} (\mathcal{D}_\pi \xmapsto{\text{m}} \mathcal{D}_\pi), \vec{\sqsubseteq}\right) \underset{\dot{\vec{\alpha}}_\pi}{\overset{\dot{\vec{\gamma}}_\pi}{\leftrightarrows}} \left(\left(\prod_{\pi' \in \Gamma[\![\pi]\!]} \mathcal{D}^\sharp_{\pi'}\right) \xmapsto{\text{m}} (\mathcal{D}^\sharp_\pi \xmapsto{\text{m}} \mathcal{D}^\sharp_\pi), \vec{\sqsubseteq}^\sharp\right)$$

we define:

$$\dot{\vec{\alpha}}_\pi \stackrel{\text{def}}{=} \lambda F\boldsymbol{\cdot}\, \vec{\alpha}_\pi \circ F \circ \dot{\gamma}_\pi \qquad \dot{\vec{\gamma}}_\pi \stackrel{\text{def}}{=} \lambda F^\sharp\boldsymbol{\cdot}\, \vec{\gamma}_\pi \circ F^\sharp \circ \dot{\alpha}_\pi$$

It follows that the semantic transformers $\mathcal{F}_\pi$, $\pi \in \Gamma^\star [\![\Pi]\!]$ associated with the program Π can be abstracted compositionally into $\mathcal{F}^\sharp_\pi$, $\pi \in \Gamma^\star [\![\Pi]\!]$, by induction on the program syntax:

Definition 25. The abstract semantic transformers $\mathcal{F}^\sharp_\pi$ associated with components $\pi \in \Gamma^\star [\![\Pi]\!]$ of program $\Pi \in \mathcal{P}$ are defined such that:

– for atomic component $\pi \in \Gamma^\star_a [\![\Pi]\!]$ of Π, $\mathcal{F}^\sharp_\pi \overset{\text{def}}{=} \vec{\alpha}_\pi(\mathcal{F}_\pi)$;

– for compound components $\pi \in \Gamma^\star_c [\![\Pi]\!]$ of Π, $\mathcal{F}^\sharp_\pi \overset{\text{def}}{=} \dot{\vec{\alpha}}_\pi(\mathcal{F}_\pi)$. (6)

Observe that the abstractions of Hyp. 23 completely determine the semantic transformers for all program subcomponents. We will show that this construction ensures the soundness of the abstraction. For completeness, we say that:

Definition 26. The abstraction α is *exact*, *faithful* or *complete* if and only if for all $\pi \in \Gamma^\star [\![\Pi]\!]$ and $\Phi = \prod_{\pi' \in \Gamma [\![\pi]\!]} \mathcal{S}_{\text{fi}} [\![\pi']\!]$:

$$\begin{cases} \alpha_\pi \circ \mathcal{F}_\pi \;=\; \mathcal{F}^\sharp_\pi \circ \alpha_\pi & \text{if } \pi \in \Gamma^\star_a [\![\Pi]\!] \\ \alpha_\pi \circ \mathcal{F}_\pi(\Phi) \;=\; \mathcal{F}^\sharp_\pi(\dot{\alpha}_\pi(\Phi)) \circ \alpha_\pi & \text{if } \pi \in \Gamma^\star_c [\![\Pi]\!] \end{cases}$$

8.4 Abstract semantic definitions

The abstract semantics is defined compositionally, by induction on the syntax of the program Π in the same way as the concrete semantics:

Definition 27. $\mathcal{S}^\sharp_{\text{fi}} [\![\Pi]\!]$ is defined as in Def. 4 (respectively $\mathcal{S}^\sharp_{\text{eq}} [\![\Pi]\!]$ as in Def. 6, $\mathcal{S}^\sharp_{\text{co}} [\![\Pi]\!]$ as in Def. 8, $\mathcal{S}^\sharp_{\text{cl}} [\![\Pi]\!]$ as in Def. 12, $\mathcal{S}^\sharp_{\text{ru}} [\![\Pi]\!]$ as in Def. 15 and $\mathcal{S}^\sharp_{\text{ga}} [\![\Pi]\!]$ as in Def. 19) using the abstract semantic transformers $\mathcal{F}^\sharp_\pi$ in place of the concrete transformers $\mathcal{F}_\pi$, $\pi \in \Gamma^\star [\![\Pi]\!]$.

For all program subcomponents, the abstractions of Hyp. 23 ensure the soundness of the abstract semantics Def. 27:

Proposition 28. *Let $\mathcal{S} [\![\cdot]\!]$ be either $\mathcal{S}_{\text{fi}} [\![\cdot]\!]$, $\mathcal{S}_{\text{eq}} [\![\cdot]\!]$, $\mathcal{S}_{\text{co}} [\![\cdot]\!]$, $\mathcal{S}_{\text{cl}} [\![\cdot]\!]$, $\mathcal{S}_{\text{ru}} [\![\cdot]\!]$ or $\mathcal{S}_{\text{ga}} [\![\cdot]\!]$ and $\mathcal{S}^\sharp [\![\cdot]\!]$ be respectively either $\mathcal{S}^\sharp_{\text{fi}} [\![\cdot]\!]$, $\mathcal{S}^\sharp_{\text{eq}} [\![\cdot]\!]$, $\mathcal{S}^\sharp_{\text{co}} [\![\cdot]\!]$, $\mathcal{S}^\sharp_{\text{cl}} [\![\cdot]\!]$, $\mathcal{S}^\sharp_{\text{ru}} [\![\cdot]\!]$ or $\mathcal{S}^\sharp_{\text{ga}} [\![\cdot]\!]$. The abstract semantics $\mathcal{S}^\sharp [\![\pi]\!]$ is sound i.e. for all $\pi \in \Gamma^\star [\![\Pi]\!]$, we have $\mathcal{S}^\sharp [\![\pi]\!] \sqsubseteq^\sharp \alpha_\pi(\mathcal{S} [\![\pi]\!])$.*

Proposition 29. *Moreover, if the abstraction is complete, then the abstract semantics $\mathcal{S}^\sharp [\![\pi]\!]$ is also complete i.e. for all $\pi \in \Gamma^\star [\![\Pi]\!]$, we have $\mathcal{S}^\sharp [\![\pi]\!] = \alpha_\pi(\mathcal{S} [\![\pi]\!])$.*

9. Example

As a very simple example of abstraction, we consider the collecting of letters occurring in the sentences of a language and apply it to approximate the semantics of μ-expressions. This illustrates the formal compositional derivation of the abstract semantics from its definition.

The theory of abstract interpretation provides various ways of approximating each constructor (sum, lift, (smash) product, function space, powerset/domain, etc.) of set/domain theory [6, 9]. Since semantic domains are defined inductively using these constructors, abstraction can be lifted compositionally to abstract semantic domains, in general by induction on the rank (measuring the complexity) of the semantic domain. For example, in the case of μ-expressions, the basis is given by the abstraction of a language L by the set of letters x appearing in sentences σ of L. $\alpha_v \in \wp(\mathbb{S}) \mapsto \wp(\mathbb{A})$ is defined as $\alpha_v(\emptyset) \stackrel{\text{def}}{=} \emptyset$, $\alpha_v(L) \stackrel{\text{def}}{=} \bigcup_{\sigma \in L} \alpha_s(\sigma)$ where $\alpha_s(a) \stackrel{\text{def}}{=} \{a\}$ and $\alpha_s(a\sigma) \stackrel{\text{def}}{=} \{a\} \cup \alpha_s(\sigma)$. Let us define the abstract value domain $\mathbb{V}^\sharp \stackrel{\text{def}}{=} \wp(\mathbb{A})$. Since α_v is a complete $\cup$-morphism, there exists a unique γ_v such that

$$(\mathbb{V}, \subseteq) \underset{\alpha_v}{\overset{\gamma_v}{\leftrightarrows}} (\mathbb{V}^\sharp, \subseteq) \tag{7}$$

is a Galois connection. Defining abstract environments $\mathbb{E}^\sharp \stackrel{\text{def}}{=} \mathsf{V} \mapsto \mathbb{V}^\sharp$ and the pointwise abstraction:

$$\alpha_e(\rho) \stackrel{\text{def}}{=} \lambda X \boldsymbol{\cdot} \alpha_v(\rho(X)) \qquad \gamma_e(\rho^\sharp) \stackrel{\text{def}}{=} \lambda X \boldsymbol{\cdot} \gamma_v(\rho^\sharp(X)) \tag{8}$$

we get the Galois connection:

$$(\mathbb{E}, \dot{\subseteq}) \underset{\alpha_e}{\overset{\gamma_e}{\leftrightarrows}} (\mathbb{E}^\sharp, \dot{\subseteq}) \tag{9}$$

The abstract semantic domain $\mathbb{D}^\sharp \stackrel{\text{def}}{=} \mathbb{E}^\sharp \mapsto \mathbb{V}^\sharp$ is defined with the functional abstraction:

$$\alpha(\varphi) \stackrel{\text{def}}{=} \alpha_v \circ \varphi \circ \gamma_e \qquad \gamma(\varphi^\sharp) \stackrel{\text{def}}{=} \gamma_v \circ \varphi^\sharp \circ \alpha_e \tag{10}$$

which is a Galois connection $(\mathbb{D}, \dot{\subseteq}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\mathbb{D}^\sharp, \dot{\subseteq})$. The abstract atomic transformer domain is $\mathbb{T}^\sharp_a \stackrel{\text{def}}{=} \mathbb{D}^\sharp \stackrel{\text{m}}{\longmapsto} \mathbb{D}^\sharp$. The correspondence with the concrete atomic transformer domain is defined by the functional abstraction

$$\vec{\alpha}(\varphi) \stackrel{\text{def}}{=} \alpha \circ \varphi \circ \gamma \qquad \vec{\gamma}(\varphi^\sharp) \stackrel{\text{def}}{=} \gamma \circ \varphi^\sharp \circ \alpha \tag{11}$$

This is a Galois connection $(\mathbb{T}_a, \vec{\subseteq}) \underset{\vec{\alpha}}{\overset{\vec{\gamma}}{\leftrightarrows}} (\mathbb{T}^\sharp_a, \vec{\subseteq})$. Finally, the abstract compound transformer domains are $\mathbb{T}^{\sharp\,n}_c \stackrel{\text{def}}{=} \mathbb{D}^{\sharp\,n} \stackrel{\text{m}}{\longmapsto} \mathbb{T}^\sharp_a$, for $n > 0$. The correspondence with the concrete transformer domains is defined by the functional abstraction:

$$\dot{\vec{\alpha}}^n(\Phi) \stackrel{\text{def}}{=} \vec{\alpha} \circ \Phi \circ \dot{\gamma}^n \qquad \dot{\vec{\gamma}}^n(\Phi^\sharp) \stackrel{\text{def}}{=} \vec{\gamma} \circ \Phi^\sharp \circ \dot{\alpha}^n \tag{12}$$

where:

$$\dot{\varphi}^n(\langle x_1, \ldots x_n \rangle) \stackrel{\text{def}}{=} \langle \varphi(x_1), \ldots \varphi(x_n) \rangle \tag{13}$$

This is a Galois connection $(\mathbb{T}_c{}^n, \dot{\vec{\subseteq}}) \underset{\dot{\vec{\alpha}}^n}{\overset{\dot{\vec{\gamma}}^n}{\leftrightarrows}} (\mathbb{T}^{\sharp\,n}_c, \dot{\vec{\subseteq}})$.

Prop. 28 provides the foundation for designing the abstract semantics compositionally. The only remaining work consists in designing the abstract predicate transformers.

The abstract predicate transformers can be derived formally from their specification Def. 25 by algebraic computation. We illustrate this derivation for:

$$\begin{aligned}
&\mathcal{F}^\sharp_{\mu X \boldsymbol{\cdot} E} \in \mathbb{T}^{\sharp\,1}_c \\
&\stackrel{\text{def}}{=} \dot{\vec{\alpha}}^1(\mathcal{F}_{\mu X \boldsymbol{\cdot} E}) && \text{by definition (6)} \\
&= \vec{\alpha} \circ \mathcal{F}_{\mu X \boldsymbol{\cdot} E} \circ \dot{\gamma}^1 && \text{by definition (12) of } \dot{\vec{\alpha}}^1
\end{aligned}$$

$$
\begin{aligned}
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \vec{\alpha}(\mathcal{F}_{\mu X\cdot\, E}(\dot{\gamma}^1(\langle\varphi^\sharp\rangle))) && \text{since } \varphi \circ \psi = \lambda x\cdot\, \varphi(\psi(x)) \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \vec{\alpha}(\mathcal{F}_{\mu X\cdot\, E})(\langle\gamma(\varphi^\sharp)\rangle)) && \text{by definition (13) of } \dot{\gamma}^1 \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \vec{\alpha}(\lambda\langle\varphi\rangle\cdot\, \lambda\mathcal{X}\cdot\, \lambda\rho\cdot\, \varphi(\rho[X := \mathcal{X}(\rho)])(\langle\gamma(\varphi^\sharp)\rangle)) && \text{by def. (5) of } \mathcal{F}_{\mu X\cdot\, E} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \vec{\alpha}(\lambda\mathcal{X}\cdot\, \lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \mathcal{X}(\rho)])) && \text{since } \lambda\langle x\rangle\cdot\, e_1(\langle e_2\rangle) = e_1[x := e_2] \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \alpha \circ \lambda\mathcal{X}\cdot\, \lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \mathcal{X}(\rho)]) \circ \gamma && \text{by definition (11) of } \vec{\alpha} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \alpha(\lambda\mathcal{X}\cdot\, \lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \mathcal{X}(\rho)])(\gamma(\mathcal{X}^\sharp))) && \text{by } \varphi \circ \psi = \lambda x\cdot\, \varphi(\psi(x)) \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \alpha(\lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \gamma(\mathcal{X}^\sharp)(\rho)])) && \text{since } \lambda x\cdot\, e_1(e_2) = e_1[x := e_2] \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \alpha_v \circ \lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \gamma(\mathcal{X}^\sharp)(\rho)]) \circ \gamma_e && \text{by definition (10) of } \alpha \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \alpha_v(\lambda\rho\cdot\, \gamma(\varphi^\sharp)(\rho[X := \gamma(\mathcal{X}^\sharp)(\rho)])(\gamma_e(\rho^\sharp))) \\
& && \text{since } \varphi \circ \psi = \lambda x\cdot\, \varphi(\psi(x)) \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \alpha_v(\gamma(\varphi^\sharp)(\gamma_e(\rho^\sharp)[X := \gamma(\mathcal{X}^\sharp)(\gamma_e(\rho^\sharp))])) \\
& && \text{since } \lambda\langle x\rangle\cdot\, e_1(\langle e_2\rangle) = e_1[x := e_2] \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \alpha_v(\gamma_v \circ \varphi^\sharp \circ \alpha_e(\gamma_e(\rho^\sharp)[X := \gamma(\mathcal{X}^\sharp)(\gamma_e(\rho^\sharp))])) && \text{by def. (10) of } \gamma \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\alpha_e(\gamma_e(\rho^\sharp)[X := \gamma(\mathcal{X}^\sharp)(\gamma_e(\rho^\sharp))])) \\
& && \text{since } \alpha_v \text{ is surjective so } \alpha_v \circ \gamma_v = 1 \text{ by (7)} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\alpha_e(\gamma_e(\rho^\sharp)[X := \gamma_v \circ \mathcal{X}^\sharp \circ \alpha_e(\gamma_e(\rho^\sharp))])) && \text{by def. (10) of } \gamma \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\alpha_e(\gamma_e(\rho^\sharp)[X := \gamma_v \circ \mathcal{X}^\sharp(\rho^\sharp)])) \\
& && \text{since } \alpha_e \text{ is surjective so } \alpha_e \circ \gamma_e = 1 \text{ by (9)} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\alpha_e(\lambda Z\cdot\, \gamma_v(\rho^\sharp(Z))[X := \gamma_v \circ \mathcal{X}^\sharp(\rho^\sharp)])) && \text{by def. (8) of } \gamma_e \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\alpha_e(\lambda Z\cdot\boldsymbol{(}Z = X \;?\; \gamma_v \circ \mathcal{X}^\sharp(\rho^\sharp) : \gamma_v(\rho^\sharp(Z))\boldsymbol{)})) \\
& && \text{since } \lambda Z\cdot\, \kappa(Z)[X := v] = \lambda Z\cdot\boldsymbol{(}Z = X \;?\; v : \kappa(Z)\boldsymbol{)} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\lambda Y\cdot\, \alpha_v(\lambda Z\cdot\boldsymbol{(}Z = X \;?\; \gamma_v \circ \mathcal{X}^\sharp(\rho^\sharp) : \gamma_v(\rho^\sharp(Z))\boldsymbol{)})(Y))) \\
& && \text{by definition (8) of } \alpha_e \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\lambda Y\cdot\, \alpha_v(\boldsymbol{(}Y = X \;?\; \gamma_v \circ \mathcal{X}^\sharp(\rho^\sharp) : \gamma_v(\rho^\sharp(Y))\boldsymbol{)})) \\
& && \text{since } \lambda x\cdot\, e_1(e_2) = e_1[x := e_2] \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\lambda Y\cdot\, \alpha_v \circ \gamma_v(\boldsymbol{(}Y = X \;?\; \mathcal{X}^\sharp(\rho^\sharp) : \rho^\sharp(Y)\boldsymbol{)})) \\
& && \text{since } \boldsymbol{(}b \;?\; \varphi(e_1) : \varphi(e_2)\boldsymbol{)} = \varphi(\boldsymbol{(}b \;?\; e_1 : e_2\boldsymbol{)}) \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\lambda Y\cdot\boldsymbol{(}Y = X \;?\; \mathcal{X}^\sharp(\rho^\sharp) : \rho^\sharp(Y)\boldsymbol{)}) \\
& && \text{since } \alpha_v \text{ is surjective so } \alpha_v \circ \gamma_v = 1 \text{ by (7)} \\
&= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\rho^\sharp[X := \mathcal{X}^\sharp(\rho^\sharp)]) \\
& && \text{since } \lambda Y\cdot\boldsymbol{(}Y = X \;?\; v : \kappa(Y)\boldsymbol{)} = \kappa[X := v]
\end{aligned}
$$

The other abstract semantic transformers are obtained constructively in the same way:

$$
\begin{aligned}
\mathcal{F}^\sharp_0 &= \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\{0\} & \mathcal{F}^\sharp_{E_1+E_2} &= \lambda\langle\varphi^\sharp,\ \psi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \varphi^\sharp \,\dot{\cup}\, \psi^\sharp \\
\mathcal{F}^\sharp_X &= \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \rho^\sharp(X) & \mathcal{F}^\sharp_{\mu X\cdot\, E} &= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\, \varphi^\sharp(\rho^\sharp[X := \mathcal{X}^\sharp(\rho^\sharp)]) \\
\mathcal{F}^\sharp_{aE} &= \lambda\langle\varphi^\sharp\rangle\cdot\, \lambda\mathcal{X}^\sharp\cdot\, \lambda\rho^\sharp\cdot\{a\} \,\dot{\cup}\, \varphi^\sharp
\end{aligned}
$$

It should be noted that the method for designing the abstract semantics is systematic as opposed to empirical conception with a posteriori verification of the soundness.

The abstract fixpoint semantics $\mathcal{S}^\sharp_{\text{fi}}[\![E]\!]$ of μ-expression E is:

$$\begin{array}{ll}
\mathcal{S}^\sharp_{\text{fi}}[\![0]\!] \stackrel{\text{def}}{=} \lambda\rho^\sharp\cdot\{0\} & \mathcal{S}^\sharp_{\text{fi}}[\![E_1+E_2]\!] \stackrel{\text{def}}{=} \mathcal{S}^\sharp_{\text{fi}}[\![E_1]\!] \,\dot{\cup}\, \mathcal{S}^\sharp_{\text{fi}}[\![E_2]\!] \\
\mathcal{S}^\sharp_{\text{fi}}[\![X]\!] \stackrel{\text{def}}{=} \lambda\rho^\sharp\cdot\rho^\sharp(X) & \mathcal{S}^\sharp_{\text{fi}}[\![\mu X\cdot E]\!] \stackrel{\text{def}}{=} \text{lfp}\,\lambda\mathcal{X}^\sharp\cdot\lambda\rho^\sharp\cdot\mathcal{S}^\sharp_{\text{fi}}[\![E]\!]\rho^\sharp[X := \mathcal{X}^\sharp(\rho^\sharp)] \\
\mathcal{S}^\sharp_{\text{fi}}[\![aE]\!] \stackrel{\text{def}}{=} \lambda\rho^\sharp\cdot\{a\}\,\dot{\cup}\,\mathcal{S}^\sharp_{\text{fi}}[\![E]\!] &
\end{array}$$

The equivalent abstract equational semantics $\mathcal{S}^\sharp_{\text{eq}}[\![E]\!]$ of μ-expression E is the $\dot{\subseteq}$-least solution to the system of equations:

$$\begin{array}{ll}
[\![0]\!]^\sharp = \lambda\rho^\sharp\cdot\{0\} & [\![E_1+E_2]\!]^\sharp = [\![E_1]\!]^\sharp\,\dot{\cup}\,[\![E_2]\!]^\sharp \\
[\![X]\!]^\sharp = \lambda\rho^\sharp\cdot\rho^\sharp(X) & [\![\mu X\cdot E]\!]^\sharp = \lambda\rho^\sharp\cdot[\![E]\!]^\sharp\rho^\sharp[X := [\![\mu X\cdot E]\!]^\sharp(\rho^\sharp)] \\
[\![aE]\!]^\sharp = \lambda\rho^\sharp\cdot\{a\}\,\dot{\cup}\,[\![E]\!]^\sharp &
\end{array}$$

The equivalent abstract constraint-based semantics $\mathcal{S}^\sharp_{\text{co}}[\![E]\!]$ of μ-expression E is the $\dot{\subseteq}$-least solution to the system of constraints:

$$\begin{array}{rr}
0 \in [\![0]\!]^\sharp\rho^\sharp & [\![E_1]\!]^\sharp \,\dot{\subseteq}\, [\![E_1+E_2]\!]^\sharp \\
\rho^\sharp(X) \subseteq [\![X]\!]^\sharp\rho^\sharp & [\![E_2]\!]^\sharp \,\dot{\subseteq}\, [\![E_1+E_2]\!]^\sharp \\
a \in [\![aE]\!]^\sharp\rho^\sharp & [\![E]\!]^\sharp\rho^\sharp[X := [\![\mu X\cdot E]\!]^\sharp(\rho^\sharp)] \subseteq [\![\mu X\cdot E]\!]^\sharp\rho^\sharp \\
[\![E]\!]^\sharp \,\dot{\subseteq}\, [\![aE]\!]^\sharp &
\end{array}$$

The equivalent abstract closure-condition-based semantics $\mathcal{S}^\sharp_{\text{cl}}[\![E]\!]$ of μ-expression E is the $\dot{\subseteq}$-least solution to the closure-conditions:

$$\begin{array}{rr}
0 \in [\![0]\!]^\sharp\rho^\sharp & x \in [\![E_1]\!]^\sharp\rho^\sharp \Rightarrow x \in [\![E_1+E_2]\!]^\sharp\rho^\sharp \\
x \in \rho^\sharp(X) \Rightarrow x \in [\![X]\!]^\sharp\rho^\sharp & x \in [\![E_2]\!]^\sharp\rho^\sharp \Rightarrow x \in [\![E_1+E_2]\!]^\sharp\rho^\sharp \\
a \in [\![aE]\!]^\sharp\rho^\sharp & x \in [\![E]\!]^\sharp\rho^\sharp[X := [\![\mu X\cdot E]\!]^\sharp(\rho^\sharp)] \Rightarrow x \in [\![\mu X\cdot E]\!]^\sharp\rho^\sharp \\
x \in [\![E]\!]^\sharp\rho^\sharp \Rightarrow x \in [\![aE]\!]^\sharp\rho^\sharp &
\end{array}$$

The equivalent abstract rule-based semantics $\mathcal{S}^\sharp_{\text{ru}}[\![E]\!]$ of μ-expression E is defined by the following formal system:

$$0 \in \mathcal{S}^\sharp_{\text{ru}}[\![0]\!]\rho^\sharp \qquad \frac{x \in \rho^\sharp(X)}{x \in \mathcal{S}^\sharp_{\text{ru}}[\![X]\!]\rho^\sharp} \qquad a \in \mathcal{S}^\sharp_{\text{ru}}[\![aE]\!]\rho^\sharp \qquad \frac{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E]\!]\rho^\sharp}{x \in \mathcal{S}^\sharp_{\text{ru}}[\![aE]\!]\rho^\sharp}$$

$$\frac{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E_1]\!]\rho^\sharp}{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E_1+E_2]\!]\rho^\sharp} \qquad \frac{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E_2]\!]\rho^\sharp}{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E_1+E_2]\!]\rho^\sharp} \qquad \frac{x \in \mathcal{S}^\sharp_{\text{ru}}[\![E]\!]\rho^\sharp[X := \mathcal{S}^\sharp_{\text{ru}}[\![\mu X\cdot E]\!](\rho^\sharp)]}{x \in \mathcal{S}^\sharp_{\text{ru}}[\![\mu X\cdot E]\!]\rho^\sharp}$$

Finally, the game-theoretic abstract semantics $\mathcal{S}^\sharp_{\text{ga}}[\![E]\!]$ of μ-expression E is defined by the rules:

$$R^\sharp_0 = \{\langle\{\lambda\rho\cdot\{0\}\},\ \emptyset\rangle\} \qquad R^\sharp_X = \{\langle\{\lambda\rho\cdot\rho(X)\},\ \emptyset\rangle\}$$

$$R^\sharp_{aE} = \{\langle\{\lambda\rho\cdot\{a\}\cup\mathcal{S}^\sharp_{\text{ga}}[\![E]\!]\rho,\ \emptyset\rangle\} \qquad R^\sharp_{E_1+E_2} = \{\langle\mathcal{S}^\sharp_{\text{ga}}[\![E_1]\!],\ \emptyset\rangle,\ \langle\mathcal{S}^\sharp_{\text{ga}}[\![E_2]\!],\ \emptyset\rangle\}$$

$$R^\sharp_{\mu X\cdot E} = \{\langle\{\lambda\rho\cdot x\},\ P\rangle : P \subseteq \mathbb{S}\ \wedge\ x \in \mathcal{S}^\sharp_{\text{ga}}[\![E]\!]\rho[X := P]\}$$

Proposition 30. *The abstraction α defined in* (10) *is complete.*

Corollary 31. *For all μ-expressions E, $\mathcal{S}^\sharp_{\text{fi}}[\![E]\!] = \mathcal{S}^\sharp_{\text{eq}}[\![E]\!] = \mathcal{S}^\sharp_{\text{co}}[\![E]\!] = \mathcal{S}^\sharp_{\text{cl}}[\![E]\!] = \mathcal{S}^\sharp_{\text{ru}}[\![E]\!] = \mathcal{S}^\sharp_{\text{ga}}[\![E]\!] = \alpha(\mathcal{S}_{\text{fi}}[\![E]\!])$.*

References

[1] S. Abramsky, R. Jagadeesan, & P. Malacaria. Full abstraction for PCF (extended abstract). *Proc. TACS'94*, LNCS 789, 1–15, 1994.

[2] P. Aczel. An introduction to inductive definitions. In J. Barwise, ed., *Handbook of Mathematical Logic*, vol. 90, 739–782. Elsevier, 1977.

[3] J. Berstel & L. Boasson. Context-free languages. In [18], ch. 2, 61–102.

[4] P. Cousot. Methods and logics for proving programs. In [18], ch. 15, 843–993.

[5] P. Cousot et R. Cousot. Static determination of dynamic properties of recursive procedures. In E. Neuhold, ed., *IFIP Conference on Formal Description of Programming Concepts*, St-Andrews, N.B., Canada, 237–277. North-Holland Pub. Co., 1977.

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

[7] P. Cousot & R. Cousot. A language independent proof of the soundness and completeness of generalized Hoare logic. *Inf. & Comp.*, 80(2):165–191, 1989.

[8] P. Cousot & R. Cousot. Inductive definitions, semantics and abstract interpretation. In *19th ACM POPL*, 83–94, 1992.

[9] P. Cousot & R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *Proc. IEEE 1994 ICCL*, 95–112, 1994.

[10] L. Damas & R. Milner. Principle type schemes for functional programs. In *9th ACM POPL*, 207–212, 1982.

[11] N. Heintze. Set-based analysis of ML programs (extended abstract). In *Proc. ACM Conf. Lisp & Func. Prog.*, 1994.

[12] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580, 583, 1969.

[13] R. Milner. A complete axiomatization for observational congruence of finite state behaviors. *Inf. & Comp.*, 81:227–247, 1989.

[14] P. D. Mosses. Denotational semantics. In [18], ch. 11, 575–631.

[15] J. Palsberg & P. O'Keefe. A type system equivalent to flow analysis. In *22th ACM POPL*, 367–378, 1995.

[16] J. Palsberg & M. I. Schwartzbach. Binding-time analysis: Abstract interpretation versus type inference. In *Proc. IEEE 1994 ICCL*, 289–298, 1994.

[17] G. D. Plotkin. A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, Aarhus University, (DK), Sept. 1981.

[18] J. van Leeuwen, ed. *Formal Models and Semantics*, vol. B of *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[19] M. Wirsing. Algebraic specifications. In [18], ch. 13, 675–788.

This article appeared in:

Lecture Notes in Computer Science 939
Pierre Wolper (Ed.)
Computer Aided Verification
7th International Conference, CAV '95
Liège, Belgium, July 3 - 5, 1995
Proceedings, pages 293–308
Springer, Berlin, 1995