Galois Connection Based Abstract Interpretations for Strictness Analysis

Patrick Cousot[1] and Radhia Cousot[2]

[1] LIENS, École Normale Supérieure
45, rue d'Ulm
75230 Paris cedex 05 (France)
cousot@dmi.ens.fr

[2] LIX, École Polytechnique
91128 Palaiseau cedex (France)
radhia@polytechnique.fr

Abstract. The abstract interpretation framework based upon the approximation of a fixpoint collecting semantics using Galois connections and widening/narrowing operators on complete lattices [CC77a, CC79b] has been considered difficult to apply to Mycroft's strictness analysis [Myc80, Myc81] for which denotational semantics was though to be more adequate (because non-termination has to be taken into account), see e.g. [AH87], page 25.

Considering a non-deterministic first-order language, we show, contrary to expectation, and using the classical Galois connection-based framework, that Mycroft strictness analysis algorithm is the abstract interpretation of a relational semantics (a big-steps operational semantics including non-termination which can be defined in $G^{\infty}SOS$ either in rule-based or fixpoint style by induction on the syntax of programs [CC92])

An improved version of Johnsson's algorithm [Joh81] is obtained by a subsequent dependence-free abstraction of Mycroft's dependence-sensitive method.

Finally, a compromise between the precision of dependence-sensitive algorithms and the efficiency of dependence-free algorithms is suggested using widening operators.

Keywords: Abstract interpretation; Relational semantics; Strictness analysis; Galois connection; Dependence-free and dependence-sensitive analysis; Widening.

Abstract interpretation [CC77a, CC79b] is a method for designing approximate semantics of programs which can be used to gather information about programs in order to provide safe answers to questions about their run-time behaviours. These semantics can be used to design manual proof methods or to specify automatic program analyzers. When the semantic analysis of programs is to be automated, the answers can only be partial or approximate (that is correct/safe/sound but incomplete) since questions such as termination for all input data are undecidable.

By considering that non-terminating and erroneous behaviors are equivalent, *call-by-need* can be replaced by *call-by-value* in functional programs either whenever the actual argument is always evaluated at least once in the function body or upon the later recursive calls (so that if the evaluation of the actual argument does not terminate or is erroneous then so does the function call) or whenever the function call does

not terminate or is erroneous (whether the actual argument is evaluated or not). Alan Mycroft's *strictness analysis* [Myc80, Myc81] is an abstract interpretation designed to recognize these situations. Observe the importance of taking non-termination $\perp$ into account: left-to-right addition is strict in its second argument since it does not terminate either when evaluation of its first argument is guaranteed not to terminate (in which case the second argument is not needed) or the evaluation of its first argument does terminate but that of the second does not. In the classical definition of strictness, errors Ω must also be identified with non-termination $\perp$. Otherwise left-to-right addition would not be strict in its second argument since $\texttt{true} + \perp = (1/0) + \perp = \Omega$.

Given a recursive function declaration $f(x, y) \equiv e$, Alan Mycroft has designed an abstract function $f^\sharp(x, y) \equiv e^\sharp$ on the abstract domain $\{0, 1\}$ such that:

- if $f^\sharp(0, b) = 0$ for $b \in \{0, 1\}$ then f is strict in x;
- if $f^\sharp(a, 0) = 0$ for $a \in \{0, 1\}$ then f is strict in y;
- if $f^\sharp(1, 1) = 0$ then f never terminates;
- if $f^\sharp(a, b) = 1$ for $a, b \in \{0, 1\}$ then the abstraction is too approximate and no conclusion can be drawn.

The abstract value $e^\sharp$ of an expression e is determined as follows:

- If the expression e is reduced to a constant k then its abstract value $e^\sharp$ is 1 which is the best possible approximation to the fact that it is an integer.
- If the expression e is reduced to a formal parameter x then its evaluation is erroneous or does not terminate if and only if the evaluation of the actual argument is erroneous or does not terminate so that its abstract value $e^\sharp$ is x which denotes the value of the abstract actual argument.
- If the expression e is a basic operation such as $e_1 + e_2$ (always needing its two arguments e_1 and e_2) and the abstract values $e_1^\sharp$ and $e_2^\sharp$ of expressions e_1 and e_2 are obtained recursively, then $(e_1 + e_2)^\sharp = (e_1^\sharp \wedge e_2^\sharp)$ where $(0 \wedge 0) = (0 \wedge 1) = (1 \wedge 0) = 0$ and $(1 \wedge 1) = 1$ since the evaluation of $e_1 + e_2$ is erroneous or does not terminate when that of e_1 or that of e_2 is erroneous or does not terminate, which we conclude from $e_1^\sharp = 0$ or $e_2^\sharp = 0$.
- The evaluation of a conditional $e \equiv (e_1 \rightarrow e_2, e_3)$ is erroneous or does not terminate when the evaluation of the condition e_1 is erroneous (for example non-boolean) or does not terminate so that $e_1^\sharp = 0$ implies $e^\sharp = 0$. If the evaluation of the condition e_1 terminates and the returned boolean value is unknown (so that $e_1^\sharp = 1$), then erroneous termination or non-termination of e is guaranteed only if those of both e_2 and e_3 are guaranteed so that $e^\sharp = 0$ only if $e_2^\sharp = 0$ and $e_3^\sharp = 0$. Therefore $e^\sharp$ can be defined as $(e_1^\sharp \wedge (e_2^\sharp \vee e_3^\sharp))$ where $(0 \vee 0) = 0$ and $(0 \vee 1) = (1 \vee 0) = (1 \vee 1) = 1$.

For the traditional addition example:

$$\texttt{f(x, y)} \equiv \texttt{((x = 0)} \rightarrow \texttt{y, (1 + f(x - 1, y)))}$$

we have:

$$f^\sharp(x, y) = (x \wedge 1) \wedge (y \vee (1 \wedge f^\sharp(x \wedge 1, y)))$$

After simplifications (such as $(x \wedge 1) = (1 \wedge x) = x$, $(x \wedge 0) = (0 \wedge x) = 0$, etc.), we get:

$$f^\sharp(x, y) = x \wedge (y \vee f^\sharp(x, y))$$

so that:

$$f^\sharp(0,0) = 0 \qquad f^\sharp(1,0) = f^\sharp(1,0) \qquad f^\sharp(0,1) = 0 \qquad f^\sharp(1,1) = 1$$

Constraint $f^\sharp(0,1) = 0$ shows that f is strict in its first parameter. Constraint $f^\sharp(1,0) = f^\sharp(1,0)$ is satisfied by $f^\sharp(1,0) = 0$ proving that f is also strict in its second parameter. It is also satisfied by $f^\sharp(1,0) = 1$ which states that the call may terminate or not, which is compatible with but less precise than $f^\sharp(1,0) = 0$ stating that the call is erroneous or does not terminate.

Mycroft's strictness analysis is dependence-sensitive[3] in that for example we might have $f^\sharp(0,0) = 0$ whereas $f^\sharp(0,1) = 1$ and $f^\sharp(1,0) = 1$ so that f is jointly strict in its two parameters without being strict in any of them. Thomas Johnsson [Joh81], followed by John Hugues [Hug88], proposed a dependence-free strictness analysis method that would not discover this property. However it is more efficient and can discover useful information when no dependence relationship between parameters is needed. For the above addition example, the equations:

$$f^\sharp[\mathbf{x}](x) = x \qquad f^\sharp[\mathbf{y}](y) = y \vee f^\sharp[\mathbf{y}](y)$$

are such that $f^\sharp[\mathbf{x}](0) = 0$ and $f^\sharp[\mathbf{y}](0) = 0$ proving that f is strict in its two parameters.

The purpose of abstract interpretation is to prove the safeness of such program analysis methods with respect to a semantics, or better to formally design them by approximation of the semantics of programs.

1 Principles of Abstract Interpretation

Abstract interpretation, when reduced to its basic principles introduced in [CC77a], can be understood as the approximation of a *collecting semantics*[4].

– The collecting semantics collects a class of properties of interest about a program[5]. It is assumed to be defined as the least fixpoint $\mathrm{lfp}_\bot^\sqsubseteq F$ of a monotonic operator F where the set of *concrete properties*:

$$\mathcal{F}(\sqsubseteq, \bot, \top, \sqcup, \sqcap) \text{ is a complete lattice.} \tag{1}$$

$\sqsubseteq$ is called the *computational ordering*. The definition of the operator F is by structural induction on the syntax of the program. F is assumed to be monotonic[6] (but,

[3] In order to later avoid the confusion between "relational semantics" and "relational analysis" in the sense of S.S. Muchnick and N.D. Jones [JM81], we use "dependence-sensitive" for "relational" and "dependence-free" for "independent attribute" analysis.

[4] introduced in [CC77a] under the name *static semantics*, and renamed "collecting" by F. Nielson.

[5] invariance properties in [CC77a].

[6] $S \xrightarrow{\mathrm{m}} T$ denotes the set of total functions $\varphi \in S \mapsto T$ on posets $S(\sqsubseteq)$ and $T(\preceq)$ which are *monotonic*, that is, $\forall x, y \in S : x \sqsubseteq y \Rightarrow \varphi x \preceq \varphi y$.

for simplicity, could be assumed to be continuous[7,8]):

$$F \in \mathcal{F} \stackrel{\text{m}\sqsubseteq}{\longmapsto} \mathcal{F} \tag{2}$$

Proposition 1 Tarski's fixpoint theorem [CC79a]. *Let $\mathbb{O}$ be the class of ordinals*[9]*. Define $F^0(X) = X$, $F^{\lambda+1}(X) = F(F^\lambda(X))$ for successor ordinals $\lambda+1$ and $F^\lambda(X) = \bigsqcup_{\beta<\lambda} F^\beta(X)$ for limit ordinals λ. (1) and (2) imply that the transfinite iteration sequence $F^\lambda(\bot)$, $\lambda \in \mathbb{O}$ is well-defined, increasing for $\sqsubseteq$ and ultimately stationary $\left(\exists \epsilon \in \mathbb{O} : \forall \lambda \geq \epsilon : F^\lambda(\bot) = F^\epsilon(\bot)\right)$. Its limit:*

$$\mathrm{lfp}^{\sqsubseteq}_{\bot} F \;=\; \bigsqcup\nolimits_{\lambda \in \mathbb{O}} F^\lambda(\bot) \tag{3}$$

is the least fixpoint of F for $\sqsubseteq$. In particular if $F \in \mathcal{F} \stackrel{\text{c}\sqcup}{\longmapsto} \mathcal{F}$ is continuous then:

$$\mathrm{lfp}^{\sqsubseteq}_{\bot} F \;=\; \bigsqcup\nolimits_{n \in \mathbb{N}} F^n(\bot) \tag{4}$$

– Since the collecting semantics is not effectively computable and sometimes not even computer representable, approximations must be considered. The notion of approximation is formalized using an *approximation relation*: $\varphi \leq \phi$ means that property ϕ safely approximates φ. To effectively compute such an approximation of $\mathrm{lfp}^{\sqsubseteq}_{\bot} F$, we replace concrete properties in $\mathcal{F}$ by more crude abstract properties in a well-chosen set $\mathcal{F}^\sharp$. Concrete properties $\varphi \in \mathcal{F}$ and abstract properties $\varphi^\sharp \in \mathcal{F}^\sharp$ should be related by a "semantic function" $\gamma \in \mathcal{F}^\sharp \mapsto \mathcal{F}$. The idea is then to mimic the computation of $\mathrm{lfp}^{\sqsubseteq}_{\bot} F = \sqcup_{\lambda \in \mathbb{O}} F^\lambda(\bot)$ within $\mathcal{F}^\sharp$ as $\sqcup^\sharp_{\lambda \in \mathbb{O}} F^{\sharp\lambda}(\bot^\sharp)$ by defining $\bot^\sharp$, $F^\sharp$ and $\sqcup^\sharp$, such that:

$$\bot \leq \gamma(\bot^\sharp) \tag{5}$$

for all $\varphi \in \mathcal{F}$ and $\phi^\sharp \in \mathcal{F}^\sharp$:

$$\varphi \leq \gamma(\phi^\sharp) \;\Rightarrow\; F(\varphi) \leq \gamma(F^\sharp(\phi^\sharp)) \tag{6}$$

and for all $\{\varphi_\beta \mid \beta \in \mathbb{O}\} \subseteq \mathcal{F}$, $\{\phi^\sharp_\beta \mid \beta \in \mathbb{O}\} \subseteq \mathcal{F}^\sharp$ and all limit ordinals λ (ω only if F is continuous):

$$\left(\forall \beta \leq \beta' < \lambda : \varphi_\beta \sqsubseteq \varphi_{\beta'} \wedge \varphi_\beta \leq \gamma(\phi^\sharp_\beta)\right) \;\Rightarrow\; \bigsqcup_{\beta<\lambda} \varphi_\beta \leq \gamma\left(\bigsqcup\nolimits^\sharp_{\beta<\lambda} \phi^\sharp_\beta\right) \tag{7}$$

We get a fixpoint approximation method:

$$\mathrm{lfp}^{\sqsubseteq}_{\bot} F \leq \gamma\left(\bigsqcup\nolimits^\sharp_{\lambda \in \mathbb{O}} F^{\sharp\lambda}(\bot^\sharp)\right) \tag{8}$$

---

[7] If $\lim \in \wp(P) \rightarrowtail P$ then a mapping $\varphi \in P \mapsto P$ is said to be a *complete* lim-*morphism*, written $\varphi \in P \stackrel{\lim}{\longmapsto} P$, (respectively lim-*upper-continuous*, lim-*lower-continuous*) if $\lim\{\varphi(x) \mid x \in X\} = \varphi(\lim\{x \mid x \in X\})$ for all $X \subseteq P$ (respectively increasing and decreasing chains $X \subseteq P$) when the limits are well-defined. For short, it is *continuous* if $\sqcup$-upper-continuous.

[8] Continuity would require the non-determinism of the primitives of the functional language considered in section 2 to be bounded.

[9] Ordinals constitute an extension into the infinite of the order properties of the naturals numbers: 0, 1, 2, ..., ω, $\omega+1$, $\omega+2$, ..., $\omega.2$, $\omega.2+1$, $\omega.2+2$, ..., ω^2, ..., ω^ω, ..., $\omega^{\omega^{\cdots}}$, ... where successor ordinals have the form $\lambda+1$, $\lambda \in \mathbb{O}$ and limit ordinals are of the form $\omega.\lambda$, $\lambda \in \mathbb{O} - \{0\}$ and ω is the least upper bound of the naturals.

Proposition 2 Fixpoint approximation. *(1), (2), (5), (6) and (7) imply (8).*

– It is often the case that the approximation relation is a partial ordering:

$$\leq \text{ is a partial order on } \mathcal{F} \tag{9}$$

Then, we can consider an abstract version $\leq^\sharp$ on $\mathcal{F}^\sharp$ of the concrete approximation ordering $\leq$ on $\mathcal{F}$:[10]

$$\leq^\sharp \text{ is a partial order on } \mathcal{F}^\sharp \tag{10}$$

The abstract approximation ordering approximates the concrete one, hence the semantic function γ is monotonic:

$$\gamma \in \mathcal{F}^\sharp \stackrel{\text{m}\leq^\sharp}{\longmapsto} \mathcal{F} \tag{11}$$

An interesting situation is when all concrete properties $\varphi \in \mathcal{F}$ have a best corresponding approximation $\alpha(\varphi) \in \mathcal{F}^\sharp$ where $\alpha \in \mathcal{F} \mapsto \mathcal{F}^\sharp$. Since $\alpha(\varphi)$ is an approximation that correctly describes φ, we have:

$$\forall \varphi \in \mathcal{F} : \varphi \leq \gamma\big(\alpha(\varphi)\big) \tag{12}$$

and since it is the most precise one, we also have:

$$\forall \varphi \in \mathcal{F} : \forall \phi^\sharp \in \mathcal{F}^\sharp : \varphi \leq \gamma(\phi^\sharp) \;\Rightarrow\; \alpha(\varphi) \leq^\sharp \phi^\sharp \tag{13}$$

(9) and (10) imply that (11), (12) and (13) are equivalent to the characteristic property of *Galois connections* written:

$$\mathcal{F}(\leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{F}^\sharp(\leq^\sharp) \stackrel{\text{def}}{=} \forall \varphi \in \mathcal{F} : \forall \phi^\sharp \in \mathcal{F}^\sharp : \varphi \leq \gamma(\phi^\sharp) \;\Leftrightarrow\; \alpha(\varphi) \leq^\sharp \phi^\sharp \tag{14}$$

As suggested in [CC77c], a Galois connection can be lifted to higher-order functional spaces for the pointwise ordering:

$$\varphi \leq \phi \stackrel{\text{def}}{=} \forall x : \varphi(x) \leq \phi(x) \tag{15}$$

as follows:

Proposition 3 Functional Galois connection. *(9), (10), (14) for $\langle \alpha_0, \gamma_0 \rangle$ and $\langle \alpha_1, \gamma_1 \rangle$ and (15) for $\leq$ and $\leq^\sharp$ imply:*

$$\mathcal{F}_0 \stackrel{\text{m}\leq}{\longmapsto} \mathcal{F}_1(\leq) \underset{\lambda\Phi\cdot\alpha_1\circ\Phi\circ\gamma_0}{\overset{\lambda\Psi\cdot\gamma_1\circ\Psi\circ\alpha_0}{\rightleftarrows}} \mathcal{F}_0^\sharp \stackrel{\text{m}\leq^\sharp}{\longmapsto} \mathcal{F}_1^\sharp(\leq^\sharp) \tag{16}$$

– When no abstract property is useless, that is the abstraction function α is surjective or equivalently $\alpha \circ \gamma$ is the identity, we have a *Galois surjection* written:

$$\mathcal{F}(\leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{F}^\sharp(\leq^\sharp) \stackrel{\text{def}}{=} \mathcal{F}(\leq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{F}^\sharp(\leq^\sharp) \;\wedge\; \forall \phi \in \mathcal{F}^\sharp : \alpha \circ \gamma(\phi) = \phi \tag{17}$$

Corollary 4 Functional Galois connection. *(9), (10), (14) for $\langle \alpha_1, \gamma_1 \rangle$, (17) for $\langle \alpha_0, \gamma_0 \rangle$ and (15) for $\leq$ and $\leq^\sharp$ imply:*

$$\mathcal{F}_0 \stackrel{\text{m}\leq}{\longmapsto} \mathcal{F}_1(\leq) \underset{\lambda\Phi\cdot\alpha_1\circ\Phi\circ\gamma_0}{\overset{\lambda\Psi\cdot\gamma_1\circ\Psi\circ\alpha_0}{\rightleftarrows}} \mathcal{F}_0^\sharp \longmapsto \mathcal{F}_1^\sharp(\leq^\sharp) \tag{18}$$

Corollary 5 Functional Galois surjection. *(9), (10), (17) for $\langle \alpha_0, \gamma_0 \rangle$ and $\langle \alpha_1, \gamma_1 \rangle$ and (15) for $\leq$ and $\leq^\sharp$ imply:*

$$\mathcal{F}_0 \stackrel{\text{m}\leq}{\longmapsto} \mathcal{F}_1(\leq) \underset{\lambda\Phi\cdot\alpha_1\circ\Phi\circ\gamma_0}{\overset{\lambda\Psi\cdot\gamma_1\circ\Psi\circ\alpha_0}{\rightleftarrows}} \mathcal{F}_0^\sharp \stackrel{\text{m}\leq^\sharp}{\longmapsto} \mathcal{F}_1^\sharp(\leq^\sharp) \tag{19}$$

[10] We can always define $\varphi^\sharp \leq^\sharp \phi^\sharp \stackrel{\text{def}}{=} \gamma(\varphi^\sharp) \leq \gamma(\phi^\sharp)$ and reason on the equivalence classes $\mathcal{F}^\sharp/\equiv^\sharp$ where $\varphi^\sharp \equiv^\sharp \phi^\sharp \stackrel{\text{def}}{=} \varphi^\sharp \leq^\sharp \phi^\sharp \wedge \phi^\sharp \leq^\sharp \varphi^\sharp$.

– Proposition 3 and corollary 5 lead to the hypothesis that approximations are preserved by the semantic function F:

$$F \in \mathcal{F} \stackrel{\mathrm{m}}{\longmapsto} \mathcal{F} \tag{20}$$

Then, by defining $\bot^\sharp$, $F^\sharp$ and $\sqcup^\sharp$ such that:

$$\bot^\sharp \geq^\sharp \alpha(\bot) \tag{21}$$

$$F^\sharp \geq^\sharp \alpha \circ F \circ \gamma \tag{22}$$

and for all $\{\varphi_\beta \mid \beta \in \mathbb{O}\} \subseteq \mathcal{F}$, $\{\phi_\beta^\sharp \mid \beta \in \mathbb{O}\} \subseteq \mathcal{F}^\sharp$ and all limit ordinals λ (ω only if F is continuous):

$$\Big(\forall \beta \leq \beta' < \lambda : \varphi_\beta \sqsubseteq \varphi_{\beta'} \wedge \alpha(\varphi_\beta) \leq^\sharp \phi_\beta^\sharp\Big) \;\Rightarrow\; \alpha\Big(\bigsqcup_{\beta<\lambda} \varphi_\beta\Big) \leq^\sharp \bigsqcup_{\beta<\lambda}^\sharp \phi_\beta^\sharp \tag{23}$$

we obtain the following fixpoint approximation method:

Proposition 6 Fixpoint approximation. *(1), (2), (9), (10), (14), (15), (20), (21), (22) and (23) imply (8).*

– If the computational ordering has an abstract correspondent:

$$\mathcal{F}^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcup^\sharp, \sqcap^\sharp) \text{ is a complete lattice.} \tag{24}$$

such that:

$$\mathcal{F}(\sqsubseteq) \underset{\alpha^\rightarrow}{\overset{\gamma}{\leftrightarrows}} \mathcal{F}^\sharp(\sqsubseteq^\sharp) \tag{25}$$

then we can characterize $\bot^\sharp$ and $\sqcup^\sharp$, as follows:

$$\bot^\sharp = \alpha(\bot) \tag{26}$$

$$\bigsqcup^\sharp{}_{\lambda \in \mathbb{O}}\, \varphi_\lambda^\sharp = \alpha\Big(\bigsqcup_{\lambda \in \mathbb{O}} \gamma(\varphi_\lambda^\sharp)\Big) \tag{27}$$

Lemma 7. *(1), (24) and (25) imply (26) and (27).*

If the abstract computational and approximation orderings coincide then we can simplify hypothesis (23) as follows:

Lemma 8. *(1), (24), (25) and $\leq^\sharp = \sqsubseteq^\sharp$ imply (23).*

Corollary 9 Fixpoint approximation. *(1), (2), (9), (10), (14), (15), (20), (22), (24), (25) and $\leq^\sharp = \sqsubseteq^\sharp$ imply (8).*

2 Syntax and Relational Semantics of a First-order Functional Language

2.1 Syntax

In order to simplify the manipulation of tuples of parameters, we understand a tuple $(e_1, \ldots, e_i, \ldots, e_n)$ of actual arguments corresponding to formal parameters $(v_1, \ldots, v_i, \ldots, v_n)$ as a vector $\vec{e}$. This vector $\vec{e}$ is a notation for a special kind of functions from its domain $\mathrm{Dom}\,\vec{e} = \{v_1, \ldots, v_i, \ldots, v_n\}$ into the set of expressions such that $\vec{e}[v_i] = e_i$. The domain of $\vec{e}$ is left implicit in the notation $(e_1, \ldots, e_i, \ldots, e_n)$ but can be obtained from the syntactic context. The correspondence between v_i and e_i is given as usual by a positional notation so that each v_i and e_i has a rank $\mathrm{rk}(v_i) = \mathrm{rk}(e_i) = i$. We write $v \in \vec{e}$ for $v \in \mathrm{Dom}\,\vec{e}$ so that $\vec{e} = \prod_{v \in \vec{e}} \vec{e}[v]$. For uniformity, we also use a vector notation $\vec{v}$ for the tuple of formal parameters

$(v_1, \ldots, v_i, \ldots, v_n)$ with the convention that $v_i = \vec{v}[v_i]$. Finally, the same notation is used for functions, so that a program consists of function declarations $\vec{f}\vec{v} \equiv \vec{F}$, also written:

$$\prod_{f \in \vec{f}} f\vec{v} \equiv \vec{F}[f]$$

where the body $\vec{F}[f]$ of function f is an expression e depending upon the formal parameters $v \in \vec{v}$ and containing recursive calls to the functions $f \in \vec{f}$. This expression e is written according to the following syntax[11]:

e	$::= k$	constant
	$\mid v$	variable
	$\mid b\vec{e}$	basic operation
	$\mid f\vec{e}$	function call
	$\mid (e_1 \rightarrow e_2, e_3)$	conditional
$\vec{v}$	$::= (v_1, \ldots, v_n)$	tuple of formal parameters
$\vec{e}$	$::= (e_1, \ldots, e_n)$	tuple of actual arguments

2.2 Semantic Domains

Computed values belong to a given set Υ containing the booleans $\{\mathrm{tt}, \mathrm{ff}\}$ and other basic values such as natural numbers $\mathbb{N}$, integers $\mathbb{Z}$, reals $\mathbb{R}$, etc. We use $\Omega \notin \Upsilon$ to denote run-time errors and $\perp \notin \Upsilon$ to denote non-termination. Therefore saying that an expression has value Ω means that its evaluation terminates by the production of an error message. Saying that its value is $\perp$ means that its evaluation does not terminate. We define $\Upsilon^\Omega \stackrel{\text{def}}{=} \Upsilon \cup \{\Omega\}$, $\Upsilon_\perp \stackrel{\text{def}}{=} \Upsilon \cup \{\perp\}$ and $\Upsilon_\perp^\Omega \stackrel{\text{def}}{=} \Upsilon \cup \{\Omega, \perp\}$. Values ν taken by a variable v belong to $\mathcal{D}^\Re$ and values $\vec{\nu}$ taken by a tuple $\vec{v}$ of variables belong to $\vec{\mathcal{D}}^\Re$[12]:

$$\mathcal{D}^\Re \stackrel{\text{def}}{=} \Upsilon_\perp^\Omega \qquad \vec{\mathcal{D}}^\Re \stackrel{\text{def}}{=} \prod_{v \in \vec{v}} \mathcal{D}^\Re$$

Using Ω to denote run-time errors and $\perp$ to denote non-termination, we can always consider total functions over $\Upsilon_\perp^\Omega$ corresponding to partial functions over Υ. Contrary to a common practice in denotational semantics, which is explained and followed by [Mos90] (see page 599), we avoid representing errors by $\perp$ since we want to discriminate finite against infinite executions[13]. The *relational semantics* of a function declaration $f\vec{v} \equiv e$ is a relation $f^\Re \in \mathcal{F}^\Re$ where:

$$\mathcal{F}^\Re \stackrel{\text{def}}{=} \wp(\vec{\mathcal{D}}^\Re \times \mathcal{D}^\Re)$$

We write $f(\vec{\nu}) \rightsquigarrow \nu$ for $\langle \vec{\nu}, \nu \rangle \in f^\Re$ which means that the call of function f with actual arguments having values $\vec{\nu}$ may return the result ν. If this call may terminate

[11] Expressions e and tuples $\vec{e}$ of expressions have an attribute $e.\vec{v}$ (resp. $\vec{e}.\vec{v}$) which is the list of visible parameters and an attribute $e.\partial$ (resp. $\vec{e}.\partial$) which is the set of headings $f\vec{v}$ of the visible functions f which can occur free in the expression. Tuples of variables $\vec{v}$ (respectively tuples of expressions $\vec{e}$) have a domain attribute $\mathrm{Dom}\,\vec{v}$ (resp. $\mathrm{Dom}\,\vec{e}$) which is the set of variables occurring in the tuple (resp. the set of formal parameters corresponding to each expression in the tuple).

[12] $\vec{\mathcal{D}}^\Re$ is a shorthand for $\vec{\mathcal{D}}_{\vec{v}}^\Re = \prod_{v \in \mathrm{Dom}\,\vec{v}} \mathcal{D}^\Re$ where $\vec{v}$ is left context-dependent.

[13] More refinement of Ω might be useful. For example we could differentiate the class of run-time failures due to a type error (using "wrong" as in [Mil78] for the value $1/\mathrm{tt}$) from those due to partially defined well-typed functions (such as $1/0$). Each kind of error might even be described by a different error value (corresponding to different error messages).

erroneously then $f(\vec{\nu}) \rightsquigarrow \Omega$ whereas $f(\vec{\nu}) \rightsquigarrow \bot$ indicates that nontermination is possible. The relational semantics $\vec{f}^{\Re}$ of a program $\prod_{f \in \vec{f}} f\vec{v} \equiv \vec{F}[f]$ (where the expression $\vec{F}[f]$ is the body of function f) is a tuple of functions belonging to $\vec{\mathcal{F}}^{\Re}$:

$$\vec{\mathcal{F}}^{\Re} \stackrel{\text{def}}{=} \prod_{f \in \vec{f}} \mathcal{F}^{\Re}$$

An expression e can be evaluated when the relational semantics $\vec{\varphi}$ of the functions $\vec{f}$ and the value $\vec{\nu}$ of the variables $\vec{v}$ which may occur free in e are known. We write $\vec{\varphi}, \vec{\nu} \vdash e \rightsquigarrow \nu$ to mean that evaluation of e in environment $\vec{\varphi}$, $\vec{\nu}$ may return ν. In particular this evaluation returns an error if $\nu = \Omega$ and does not terminate when $\nu = \bot$. Therefore the relational semantics $e^{\Re}$ of the expression e belongs to $\mathcal{E}^{\Re}$[14]:

$$\mathcal{E}^{\Re} \stackrel{\text{def}}{=} \vec{\mathcal{F}}^{\Re} \stackrel{\text{m}\,\underline{\subseteq}}{\longmapsto} \wp(\vec{\mathcal{D}}^{\Re} \times \mathcal{D}^{\Re})$$

whereas the relational semantics $\vec{e}^{\,\Re}$ of a tuple of expressions $\vec{e}$ belongs to $\vec{\mathcal{E}}^{\Re}$[14]:

$$\vec{\mathcal{E}}^{\Re} \stackrel{\text{def}}{=} \vec{\mathcal{F}}^{\Re} \stackrel{\text{m}\,\underline{\subseteq}}{\longmapsto} \wp(\vec{\mathcal{D}}^{\Re} \times \vec{\mathcal{D}}^{\Re})$$

2.3 Rule-based Presentation of the Relational Semantics

We now present the relational semantics of expressions, by induction on their syntax and then the semantics of programs. We use the technique of bi-inductive definitions introduced in [CC92], but resort only to the intuitive understanding of the reader. $\vec{f}^{\Re}, \vec{\nu} \vdash e \rightsquigarrow \nu$ means that evaluation of expression e may return value ν in the evaluation environment specified by $\vec{f}^{\Re}$, giving the relational semantics $\vec{f}^{\Re}[f]$ of the functions f used in e, and $\vec{\nu}$ giving the value $\vec{\nu}[v]$ of the variables v which can occur free in e.

– We assume that the value $\underline{k} \in \Upsilon$ of the constant k is given. Therefore:

$$\vec{f}^{\Re}, \vec{\nu} \vdash k \rightsquigarrow \underline{k} \; + \tag{28}$$

Such a positive axiom schema $\vec{f}^{\Re}, \vec{\nu} \vdash e \rightsquigarrow \nu$ +, marked "+", describes finite behaviors in $\vec{\mathcal{F}}^{\Re} \mapsto \wp(\vec{\mathcal{D}}^{\Re} \times \Upsilon^{\Omega})$ that is terminating or erroneous evaluations of expression e for all possible values of $\vec{f}^{\Re}$, $\vec{\nu}$ and ν.

– A formal parameter v denotes the value of the corresponding actual parameter which is stored in $\vec{\nu}[v]$:

$$\vec{f}^{\Re}, \vec{\nu} \vdash v \rightsquigarrow \vec{\nu}[v] \; + \qquad \text{if } \vec{\nu}[v] \in \Upsilon^{\Omega} \tag{29}$$

If the evaluation of the actual parameter does not terminate, which is formally represented by the fact that $\vec{\nu}[v] = \bot$, and the corresponding formal parameter v is used, then this use leads to a non-terminating evaluation:

$$\vec{f}^{\Re}, \vec{\nu} \vdash v \rightsquigarrow \bot \; - \qquad \text{if } \vec{\nu}[v] = \bot \tag{30}$$

Such a negative axiom schema, marked "-", describes infinite behaviors in $\vec{\mathcal{F}}^{\Re} \mapsto \wp(\vec{\mathcal{D}}^{\Re} \times \{\bot\})$ that is non-terminating computations for all possible values of $\vec{f}^{\Re}$, $\vec{\nu}$ and v satisfying the side condition $\vec{\nu}[v] = \bot$[15].

---

[14] Once again, e and $\vec{e}$ is left context-dependent so as to avoid the precise but heavy notation $\vec{\mathcal{E}}^{\Re}_{\vec{e}} = \left(\prod_{f\,\vec{v} \in \vec{e}\,.\,\partial} \wp\left(\left(\prod_{v \in \mathrm{Dom}\,\vec{v}} \mathcal{D}^{\Re} \right) \times \mathcal{D}^{\Re} \right) \right) \stackrel{\text{m}\,\underline{\subseteq}}{\longmapsto} \wp\left(\left(\prod_{v \in \vec{e}\,.\,\vec{v}} \mathcal{D}^{\Re} \right) \times \left(\prod_{v \in \mathrm{Dom}\,\vec{e}} \mathcal{D}^{\Re} \right) \right)$.

[15] here all non-terminating computations deliver $\bot$ but in lazy languages they might also result in infinite data structures.

– The evaluation of a tuple $\vec{e}$ of actual parameters consists in evaluating each parameter $\vec{e}[v]$, $v \in \vec{e}$. This evaluation does not terminate if the evaluation of at least one component does not terminate. Else the evaluation of the tuple $\vec{e}$ of actual parameters is terminating or erroneous:

$$\frac{\forall v \in \vec{e} : \vec{f}^{\Re}, \vec{\nu} \vdash \vec{e}[v] \rightsquigarrow \vec{\nu}'[v]}{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}'}\,+ \qquad \text{if } \forall v \in \vec{e} : \vec{\nu}'[v] \in \Upsilon^{\Omega}$$

$$\frac{\forall v \in \vec{e} : \vec{f}^{\Re}, \vec{\nu} \vdash \vec{e}[v] \rightsquigarrow \vec{\nu}'[v]}{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}'}\,- \qquad \text{if } \exists v \in \vec{e} : \vec{\nu}'[v] = \bot$$

– We assume that the relational semantics of each basic operation b is specified by a total relation $b^{\Re} \in \mathbb{P}(\vec{\Upsilon}^{\Omega}_{\bot} \times \Upsilon^{\Omega}_{\bot})$ [16] (or a function $\underline{b} \in \vec{\Upsilon}^{\Omega}_{\bot} \mapsto \Upsilon^{\Omega}_{\bot}$ if the language is deterministic, in which case $b^{\Re} \stackrel{\text{def}}{=} \{\langle \vec{\nu}, \underline{b}(\vec{\nu})\rangle \mid \vec{\nu} \in \vec{\Upsilon}^{\Omega}_{\bot}\}$). We write $b(\vec{\nu}) \rightsquigarrow \nu$ for $\langle \vec{\nu}, \nu\rangle \in b^{\Re}$.

Example 1. • The integer addition is strict in its two parameters and is defined only for integers so that $\underline{\text{plus}}(\nu, \nu') \stackrel{\text{def}}{=} ((\nu = \bot \vee \nu' = \bot) \rightarrow \bot, ((\nu \notin \mathbb{Z} \vee \nu' \notin \mathbb{Z}) \rightarrow \Omega, (\nu + \nu')))$ where $+$ denotes the usual mathematical addition.

• For McCarthy's conjunction, we would have $\underline{\text{cand}}(\nu, \nu') \stackrel{\text{def}}{=} (\nu = \bot \rightarrow \bot, (\nu = \text{ff} \rightarrow \text{ff}, (\nu = \text{tt} \rightarrow ((\nu' \in \{\bot, \text{tt}, \text{ff}\} \rightarrow \nu', \Omega)), \Omega))$. Observe that the first parameter is passed by value since evaluation of the conjunction does not terminate as soon as this first parameter does not terminate whereas the second is passed by need, and used only if the first is tt.

• For the unbounded random assignment '?', which does not need its argument, we have $?(\nu) \rightsquigarrow n$ if and only if $n \in \mathbb{N}$.

• For the bounded random assignment, we have:

$$\begin{aligned}
?(\langle \nu, \nu'\rangle) \rightsquigarrow \bot \quad &\text{iff} \quad (\nu = \bot) \vee (\nu' = \bot),\\
?(\langle \nu, \nu'\rangle) \rightsquigarrow \Omega \quad &\text{iff} \quad (\nu \in \Upsilon^{\Omega} - \mathbb{Z}) \vee (\nu' \in \Upsilon^{\Omega} - \mathbb{Z}) \vee\\
&\qquad\qquad (\nu \in \mathbb{Z} \wedge \nu' \in \mathbb{Z} \wedge \nu > \nu'),\\
?(\langle \nu, \nu'\rangle) \rightsquigarrow z \quad &\text{iff} \quad (\nu \in \mathbb{Z} \wedge \nu' \in \mathbb{Z} \wedge \nu \leq z \leq \nu').
\end{aligned}$$

□

The value of $b\vec{e}$ is obtained by applying b to the value of the arguments $\vec{e}$ (which may be Ω or $\bot$ without preventing the result to be in Υ when the argument is not needed):

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}' \ \wedge\ b(\vec{\nu}') \rightsquigarrow \bot}{\vec{f}^{\Re}, \vec{\nu} \vdash b\vec{e} \rightsquigarrow \bot}\,-$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}' \ \wedge\ b(\vec{\nu}') \rightsquigarrow \nu}{\vec{f}^{\Re}, \vec{\nu} \vdash b\vec{e} \rightsquigarrow \nu}\,+ \qquad \text{if } \nu \in \Upsilon^{\Omega}$$

– The evaluation of the conditional $(e_1 \rightarrow e_2, e_3)$ does not terminate when the evaluation of the condition e_1 does not terminate:

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \bot}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \bot}\,-$$

[16] We define the set $\mathbb{P}(S \times T)$ of *total binary relations* on $S \times T$ as $\{\rho \in \wp(S \times T) \mid \forall s \in S : \exists t \in T : \langle s, t\rangle \in \rho\}$ so that $\rho \in \mathbb{P}(S \times T)$ implies that for every $s \in S$ there exists at least one $t \in T$ such that the pair $\langle s, t\rangle$ belongs to ρ.

The evaluation of the conditional $(e_1 \rightarrow e_2, e_3)$ terminates with an error when the evaluation of the condition e_1 terminates without returning a boolean:

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \nu \ \wedge \ \nu \in \Upsilon^{\Omega} - \{\mathrm{tt}, \mathrm{ff}\}}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \Omega}\ +$$

If the evaluation of the condition e_1 returns a boolean b, then the value of the conditional $(e_1 \rightarrow e_2, e_3)$ is that of e_2 if b is true and that of e_3 if b is false, including the run-time error and non-termination cases:

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \mathrm{tt} \ \wedge \ \vec{f}^{\Re}, \vec{\nu} \vdash e_2 \rightsquigarrow \nu}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \nu}\ + \qquad \text{if } \nu \in \Upsilon^{\Omega}$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \mathrm{tt} \ \wedge \ \vec{f}^{\Re}, \vec{\nu} \vdash e_2 \rightsquigarrow \bot}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \bot}\ -$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \mathrm{ff} \ \wedge \ \vec{f}^{\Re}, \vec{\nu} \vdash e_3 \rightsquigarrow \nu}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \nu}\ + \qquad \text{if } \nu \in \Upsilon^{\Omega}$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash e_1 \rightsquigarrow \mathrm{ff} \ \wedge \ \vec{f}^{\Re}, \vec{\nu} \vdash e_3 \rightsquigarrow \bot}{\vec{f}^{\Re}, \vec{\nu} \vdash (e_1 \rightarrow e_2, e_3) \rightsquigarrow \bot}\ -$$

– The relational semantics of the function call $f\vec{e}$ can be defined knowing the value $\vec{\nu}$ of the free variables $\vec{v}$ which may appear within $\vec{e}$ and the relational semantics $f^{\Re} = \vec{f}^{\Re}[f]$ of the function $f \in \vec{f}$ of the program $\prod_{f \in \vec{f}} f\vec{v} \equiv \vec{F}[f]$. $f^{\Re}$ specifies a set of arguments-result pairs $f(\vec{\nu}') \rightsquigarrow \nu$. The call $f\vec{e}$ may return value ν if, given the value $\vec{\nu}'$ of the actual parameters $\vec{e}$, the function f may return that value ν, that is $f(\vec{\nu}') \rightsquigarrow \nu$. Therefore, we have:

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}' \ \wedge \ f(\vec{\nu}') \rightsquigarrow \nu}{\vec{f}^{\Re}, \vec{\nu} \vdash f\vec{e} \rightsquigarrow \nu}\ + \qquad \text{if } \nu \in \Upsilon^{\Omega} \tag{31}$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{e} \rightsquigarrow \vec{\nu}' \ \wedge \ f(\vec{\nu}') \rightsquigarrow \bot}{\vec{f}^{\Re}, \vec{\nu} \vdash f\vec{e} \rightsquigarrow \bot}\ - \tag{32}$$

Observe that in an effective implementation of the above rules (31) and (32), parameters must be passed by need since a non-terminating actual parameter $\vec{e}[v]$, such that $\vec{\nu}'[v] = \bot$, will actually affects the computation only if the corresponding formal parameter v is needed in the body $\vec{F}[f]$ of function f according to rules (29) and (30). Moreover call-by-name would be inadequate since the effective parameters are evaluated only once. However this kind of implementation detail is omitted in the mathematical description of the relational semantics where the behaviours of the actual parameters, including the non-terminating ones, can be described prior to describing the behaviour of the function call.

– The relational semantics $\vec{f}^{\Re}$ of a program $\prod_{f \in \vec{f}} f\vec{v} \equiv \vec{F}[f]$ specifies that a call of function f consists in evaluating the function body $\vec{F}[f]$ with formal parameters bound to their actual values:

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{F}[f] \rightsquigarrow \nu}{f(\vec{\nu}) \rightsquigarrow \nu}\ + \qquad \text{if } \nu \in \Upsilon^{\Omega} \tag{33}$$

$$\frac{\vec{f}^{\Re}, \vec{\nu} \vdash \vec{F}[f] \rightsquigarrow \bot}{f(\vec{\nu}) \rightsquigarrow \bot}\ - \tag{34}$$