Order Theory for Big-Step Semantics

Jérôme Vouillon

To cite this version:

Jérôme Vouillon. Order Theory for Big-Step Semantics. 2011. <hal-00782145>

HAL Id: hal-00782145
https://hal.archives-ouvertes.fr/hal-00782145

Submitted on 29 Jan 2013

Order Theory for Big-Step Semantics

Jérôme Vouillon
CNRS, UMR 7126, PPS, Univ Paris Diderot, Sorbonne Paris Cité,
F-75205 Paris, France
jerome.vouillon@pps.jussieu.fr

Abstract

We show that tools from order theory, such as Kleene fixpoint theorem, can be used to define bigstep semantics that simultaneously account for both converging and diverging behaviors of programs. These semantics remain very concrete. In particular, values are defined syntactically: the semantics of a function abstraction is a function closure rather than some abstract continuous function.

1. Introduction

Many techniques are available to express formally the semantics of a calculus. Syntactic approaches are often favored, as they are very concrete and require little theory. They can be classified into two categories. Small-step semantics [26] are very expressive, accounting for both termination and non-termination. However, this precision can be a hinder when proving program transformations: the semantics of programs are execution traces, that needs to be transformed to validate program transformations. Big-step semantics [18] are more abstract: the semantics of a program is its final outcome. They do not account for non-termination, unless they are specified as both an inductive relation for terminating programs and a coinductive relation for non-terminating ones. Denotational approaches, on the other hand, are powerful but require a significant mathematical background: the semantics of a program is not a syntactic object, but belongs to an abstract mathematical domain; for instance, function abstractions are typically interpreted as continuous functions between domains. In the present paper, we attempt to find a middle ground: we experiment with basic denotational tools to define big step semantics that simultaneously account for finite and infinite program behaviors. This way, we can define semantics which are simple and very concrete, but also precise.

We present our technique on a kind of call-by-need calculus which would be complicated to tackle by other means (Section 2). We then discuss how to apply our framework for the semantics of other calculi (Section 3). This paper is based on a mechanised formalisation in Coq of (basic) order theory and of the call-by-need calculus (Section 4).

2. A Call-by-Need Calculus

As a running example, we consider a small calculus with first-class functions and pairs. We mean this calculus to be call-by-need. Our aim is to give a kind of big-step semantics to this calculus. This semantics accounts for both converging and diverging behaviors. We then illustrate how to reason on the semantics by proving a type soundness result.

2.1 Syntax

Terms T are given by the grammar below. This is the standard syntax of λ-calculus with pairs.

$$\begin{array}{rcll} t & ::= & x & \text{variable} \\ & & \lambda x.t & \text{function abstraction} \\ & & t\,t & \text{function application} \\ & & (t,t) & \text{pair} \\ & & \mathtt{fst}\,t & \text{first projection} \\ & & \mathtt{snd}\,t & \text{second projection} \end{array}$$

We write X for the countable set of variables x. Terms evaluate to values V, described by the following grammar.

$$\begin{array}{rcll} v & ::= & (\lambda x.t)_\rho & \text{function closure} \\ & & (v,v) & \text{pair} \\ & & \mathtt{diverge} & \text{divergence} \\ & & \mathtt{error} & \text{error} \end{array}$$

An environment ρ is a total map from variables X to values V. A variable x is unbound in an environment ρ when $\rho(x) = \mathtt{error}$. The `diverge` and `error` values may occur nested inside other values: intuitively, values are computed lazily; then, when visiting a value, one may reach a position in the value where the computation diverges or fails.

Laziness makes it possible to define infinite datastructures. Thus, for the moment, we consider that values v may be infinite trees (see just below for a formal definition). In other words, we interpret the grammar above in a coinductive way. Later on, we will take a different point of view, and will find it convenient to write v to denote just finite values and $\bar{v}$ to denote any value, including infinite ones.

Infinite tree. We write $\mathbb{N}^*$ for the set of finite sequences of integers. The empty sequence is written ϵ, and $w \cdot w'$ denotes the concatenation of two sequences w and w'.

$$\textsc{Var}\quad \rho \vdash x \rightsquigarrow \rho(x) \qquad \textsc{Abs}\quad \rho \vdash \lambda x.t \rightsquigarrow (\lambda x.t)_\rho \qquad \textsc{Pair}\quad \frac{\rho \vdash t_1 \rightsquigarrow v_1 \qquad \rho \vdash t_2 \rightsquigarrow v_2}{\rho \vdash (t_1, t_2) \rightsquigarrow (v_1, v_2)}$$

$$\textsc{App}\quad \frac{\rho \vdash t \rightsquigarrow (\lambda x.t'')_{\rho'} \qquad \rho \vdash t' \rightsquigarrow v' \qquad \rho'; x \mapsto v' \vdash t'' \rightsquigarrow v}{\rho \vdash t\, t' \rightsquigarrow v} \qquad \textsc{App-Strict}\quad \frac{\rho \vdash t \rightsquigarrow \mathtt{diverge}}{\rho \vdash t\, t' \rightsquigarrow \mathtt{diverge}} \qquad \textsc{App-Error}\quad \frac{\rho \vdash t \rightsquigarrow v \qquad v \not\lesssim (\lambda x.t'')_{\rho'}}{\rho \vdash t\, t' \rightsquigarrow \mathtt{error}}$$

$$\textsc{Fst}\quad \frac{\rho \vdash t \rightsquigarrow (v_1, v_2)}{\rho \vdash \mathtt{fst}\, t \rightsquigarrow v_1} \qquad \textsc{Fst-Strict}\quad \frac{\rho \vdash t \rightsquigarrow \mathtt{diverge}}{\rho \vdash \mathtt{fst}\, t \rightsquigarrow \mathtt{diverge}} \qquad \textsc{Fst-Error}\quad \frac{\rho \vdash t \rightsquigarrow v \qquad v \not\lesssim (v_1, v_2)}{\rho \vdash \mathtt{fst}\, t \rightsquigarrow \mathtt{error}}$$

$$\textsc{Snd}\quad \frac{\rho \vdash t \rightsquigarrow (v_1, v_2)}{\rho \vdash \mathtt{snd}\, t \rightsquigarrow v_1} \qquad \textsc{Snd-Strict}\quad \frac{\rho \vdash t \rightsquigarrow \mathtt{diverge}}{\rho \vdash \mathtt{snd}\, t \rightsquigarrow \mathtt{diverge}} \qquad \textsc{Snd-Error}\quad \frac{\rho \vdash t \rightsquigarrow v \qquad v \not\lesssim (v_1, v_2)}{\rho \vdash \mathtt{snd}\, t \rightsquigarrow \mathtt{error}}$$

Figure 1. Semantics.

We assume given a set Σ of labels together with an arity function $Ar : \Sigma \to \mathbb{N} \cup \{\infty\}$. A *tree* (T, V) is a pair of a non-empty set of nodes $T \subseteq \mathbb{N}^*$ and a labelling function $V : T \to \Sigma$ such that:

- if $t \cdot c \in T$ where $c \in \mathbb{N}$, then $t \in T$ and for all $c' \in \mathbb{N}$, if $c' < c$ then $t \cdot c' \in T$;
- for all $t \in T$, $Ar(V(t)) = d(t)$, where the degree of a node $t \in T$ is $d(t) = \inf\{c \in \mathbb{N} \mid t \cdot c \notin T\}$.

We refer you to [7, 14] for more details on infinite trees.

2.2 Semantics

The tentative semantics is presented using inference rules in Figure 1. The semantics $\rho \vdash t \rightsquigarrow v$ is a relation between an environment $\rho \in X \to V$, a term $t \in T$ and the semantic value $v \in V$ of the term in the environment. As side-condition to some of the rules, we write $v \not\lesssim v'$ where v is a value and v' is the shape of a value to mean that value v is neither `diverge` nor of the shape v'. The inference rules read well. The value of a variable is taken from the environment. The value of a function abstraction is a closure composed of the abstraction together with the current environment. The value of a pair of two terms is the pair composed of the values of each of the terms. For value destructors (function application and pair projections), there are three cases. Consider a function application $t\, t'$. If the value of term t is a closure $(\lambda x.t'')_{\rho'}$, then the value of the application is the value of the closure body t'' in the environment $\rho'; x \mapsto v'$ derived from the closure environment ρ' by assigning to the function parameter x the value v' of the argument t'. If term t diverges, then so does the function application. Otherwise, the application fails. The rules for projections are similar. If the value of a term t is a pair, then the value of the projection `fst` t (resp. `snd` t) is the first element (resp. the second element) of the pair. If the term diverges, then so does the projection. Otherwise, the projection fails.

The semantics can be understood as an abstraction of call-by-need that does not account for when a term is evaluated, but only expresses that terms are evaluated at most once. Hence, we avoid the imperative flavour of typical operational semantics of call-by-need, where thunks are created to delay the evaluation of terms, and are later updated in place once their value is computed. One might rightly argue that, as is, one cannot really distinguish the calculus from a call-by-name calculus. The difference would really become visible if a non-deterministic construction were added (a non-deterministic choice, for instance, as in [19]). Indeed, consider an expression $(\lambda x.(x, x))\, t$ where subexpression t may evaluate to several distinct values due to non-determinism. In a call-by-need settings, as t is evaluated at most once, the whole expression only evaluates to values of the shape (v, v). This is reflected in our calculus by the fact that a function parameter x is bound to a single value when a function application is evaluated. On the other hand, in a call-by-name settings, the expression can also evaluate to values of the shape (v_1, v_2), where v_1 and v_2 are two distinct values of subexpression t.

Now, what do the inference rules of Figure 1 actually mean? The standard interpretation of inference rules is as *inductive* rules, that is, the assertion $\rho \vdash t \rightsquigarrow v$ holds if, using the inference rules, one can build a finite tree (a *derivation*) that justifies this assertion. An example of derivation is given in Figure 2. But we are only capturing finite behaviors this way: it is not possible to derive $\rho \vdash t \rightsquigarrow \mathtt{diverge}$ for any term t (when the environment ρ does not already contain the `diverge` value), as all rules that yield divergence also assume divergence as hypothesis! It is also not possible to derive that a term evaluates to any infinite value. Thus, a different interpretation of inductive rules is needed. For this, let us first come back to the definition of inference rules.

$$\dfrac{\overset{\textsc{Abs}}{\emptyset \vdash \lambda x.(x,x) \leadsto (\lambda x.(x,x))_\emptyset} \qquad \overset{\textsc{Abs}}{\emptyset \vdash f \leadsto (f)_\emptyset} \qquad \dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (f)_\emptyset} \quad \overset{\textsc{Var}}{\rho \vdash x \leadsto (f)_\emptyset}}{\rho \vdash (x,x) \leadsto ((f)_\emptyset,(f)_\emptyset)}\,\textsc{Pair}}{\emptyset \vdash (\lambda x.(x,x))\, f \leadsto ((f)_\emptyset,(f)_\emptyset)}\,\textsc{App}$$

where $f = \lambda x.x$ and $\rho = x \mapsto (f)_\emptyset$.

Figure 2. Simple derivation.

$$\dfrac{\overset{\textsc{Abs}}{\emptyset \vdash \Delta \leadsto (\Delta)_\emptyset} \qquad \overset{\textsc{Abs}}{\emptyset \vdash \Delta \leadsto (\Delta)_\emptyset} \qquad \dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (\Delta)_\emptyset} \quad \overset{\textsc{Var}}{\rho \vdash x \leadsto (\Delta)_\emptyset} \quad \dfrac{\vdots}{\rho \vdash x\, x \leadsto \texttt{diverge}}\,\textsc{App}}{\rho \vdash x\, x \leadsto \texttt{diverge}}\,\textsc{App}}{\emptyset \vdash \Delta\, \Delta \leadsto \texttt{diverge}}\,\textsc{App}$$

where $\rho = x \mapsto (\Delta)_\emptyset$.

Figure 3. Derivation for a diverging term.

$$\dfrac{\overset{\textsc{Abs}}{\emptyset \vdash \Pi \leadsto (\Pi)_\emptyset} \qquad \dfrac{\dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (\Pi)_\emptyset} \quad \dfrac{\vdots}{\rho \vdash (x\,x, x\,x) \leadsto p}\,\textsc{Pair}}{\rho \vdash x\,x \leadsto p}\,\textsc{App}}{\rho \vdash (x\,x, x\,x) \leadsto p}\,\textsc{Pair}}{\emptyset \vdash \Pi\,\Pi \leadsto p}\,\textsc{App}$$

where $\rho = x \mapsto (\Pi)_\emptyset$ and the value p satisfies $p = (p,p)$.

Figure 4. Term evaluating to an infinite value.

Inference system. An *inference system* Φ over a set $\mathcal{U}$ is a set of pairs $(A,c) \in \mathcal{P}(\mathcal{U}) \times \mathcal{U}$ [2]. The pair (A,c) is called an *inference rule* and is usually written:

$$\frac{a_1 \quad \dots \quad a_n}{c}$$

where $A = \{a_1, \dots, a_n\}$. The intuitive reading of such a rule is that the conclusion c holds if and only if all the antecedents in A hold.

Given an inference system Φ, we define the operator $F_\Phi : \mathcal{P}(\mathcal{U}) \to \mathcal{P}(\mathcal{U})$ as

$$F_\Phi(S) = \{c \in \mathcal{U} \mid \exists A \subseteq S, (A,c) \in \Phi\}.$$

The set $F_\Phi(S)$ is the set of conclusions that can inferred in one step from the antecedents in S. The operator F_Φ is monotonic. By Knaster-Tarski [29] theorem, it has a least and a greatest fixpoint. The least fixpoint $\operatorname{lfp} F_\Phi$ is the *inductive* interpretation of the inference system Φ. The greatest fixpoint $\operatorname{gfp} F_\Phi$ is the *coinductive* interpretation of the inference system Φ.

There are reasoning principles associated to these fixpoints. The *induction principle* can be stated as follows: if X is F-closed (that is, $F(X) \subseteq X$), then $\operatorname{lfp} F \subseteq X$. The *coinduction principle* can be stated as follows: if X is F-consistent (that is, $X \subseteq F(X)$), then $X \subseteq \operatorname{gfp} F$.

Using the coinduction principle, one can prove that an element x is included in $\operatorname{gfp} X$ by finding an F-consistent set X such that $x \in X$. In the case of the least fixpoint, one uses the fact that $\operatorname{lfp} F = \bigcup F^n(\emptyset)$. Thus, $x \in \operatorname{lfp} F$ if there exists n such that $x \in F^n(\emptyset)$.

Concretely, with the inductive interpretation, a conclusion c holds if one can build a finite tree that derives c using the inference rules in system Φ. With the coinductive interpretation, a conclusion c holds if one can build a possibly infinite tree ending with conclusion c. The F-consistent set that proves that c holds is the set of elements of $\mathcal{U}$ that occur in the derivation tree. We refer you to [14], for instance, for more details on coinductive reasoning.

In our case, the rules of Figure 1 define a map F on semantic relations, where a *semantic relation* is a relation between an environment $\rho \in X \to V$, a term $t \in T$ and the semantic value $v \in V$ of the term in the environment. We have seen that the inductive interpretation of the rules (the least fixpoint of F) is not large enough to encompass diverging behaviors. The coinductive interpretation seems more promising. For instance, one can derive that term $\Delta\,\Delta$, where $\Delta = \lambda x.x\,x$, diverges: indeed, the derivation in Figure 3 can be extended forever by repeating the same pattern of function application and pair evaluation. Similarly, one

$$\frac{\rho \leq \rho'}{(\lambda x.t)_\rho \leq (\lambda x.t)_{\rho'}} \qquad \frac{v_1 \leq v_1' \qquad v_2 \leq v_2'}{(v_1, v_2) \leq (v_1', v_2')}$$

$$\texttt{diverge} \leq v \qquad \frac{\text{for all } x,\ \rho(x) \leq \rho'(x)}{\rho \leq \rho'}$$

Figure 7. Order on finite values and environments.

can show that some terms evaluate to infinite values. Figure 4 illustrates the fact that term $\Pi\,\Pi$, where $\Pi = \lambda x.(x\,x, x\,x)$, evaluates to the infinite value p such that $p = (p, p)$.

But, actually, the derivation of Figure 3 can be generalised to show that term $\Delta\,\Delta$ evaluates to any value v, as shown in Figure 5. This is really not what we want! Thus, while the inductive interpretation of the inference rules is too small, the coinductive interpretation appears to be too large. It seems that we need to find a fixpoint in-between.

More precisely, the issue we have is that the calculus should be deterministic: to each environment ρ and term t should be associated exactly one value v. This is not the case with the fixpoints we have considered so far. But remark that operator F returns a function when given as input a function.

FACT 1. *Operator F is a self-map on the graph of functions in $(X \to V) \to T \to V$.*

In other words, the operator F preserves determinism. This gives us some confidence that it has a deterministic fixpoint. By the way, this is also a strong hint that we have the right set of rules, with no missing nor overlapping rules.

Thus, we want a fixpoint of operator F that is a function. But how to find such a fixpoint? The solution comes from order theory: we need to find a suitable domain of functions in which operator F has a least fixpoint. To find this fixpoint, the idea is to consider approximate derivations, that corresponds intuitively to giving up the computation of parts of a value at some points, returning the `diverge` value instead. Approximate derivations are partial derivations where all assertions $\rho \vdash t \leadsto v$ are justified (like for a normal finite derivation), except possibly for some assertions of the shape $\rho \vdash t \leadsto \texttt{diverge}$. An example is given in Figure 6, where the rightmost assertion at the top is not justified. Then, a correct derivation should be the limit of a suitable set of approximate derivations. For instance, the partial derivation in Figure 6 shows that the resulting value is a pair of pairs. By considering larger partial derivations, one can get a more precise approximation of the value. This approach rules out the derivation in Figure 5, where value v is coming out of this air, from infinity: if one gives up the computation at any point, one does not get any value as a result but `diverge`.

We will progressively make these ideas precise. As a first step, we define the approximation relation between *finite* values by the inductive rules in Figure 7: an approximation of a value is built by replacing some of its subvalues by the `diverge` value. Intuitively, it is then enough to work with finite values, as we will be able to approximate infinite values as precisely as wanted using only finite values (in a sense that will be made precise later). Hence, from now on, we write v to denote only finite values, and $\overline{v}$ to denote any value, including infinite ones. The inference rules of Figure 1 are reinterpreted accordingly. As we use heavily orders and more generally preorders below, we remind their definitions.

Preorder. A *preorder* $\leq$ on a set A is a binary relation on this set which is both *reflexive* (for all x in A, we have $x \leq x$) and *transitive* (for all x, y, z in A, if $x \leq y$ and $y \leq z$, then $x \leq z$). A set A with a preorder $\leq$ defined on it is called a *preordered set*. We denote it by $(A, \leq)$, or just A when there is no confusion.

An *order* is a preorder that is also *antisymmetric* (that is, if for all x, y in A, $x \leq y$ and $y \leq x$ implies $x = y$). A set A with an order $\leq$ defined on it is called an *ordered set*.

A preorder induces an equivalence relation $\equiv$ on A ($x \equiv y$ iff $x \leq y$ and $y \leq x$), and an order on the quotient set $A/{\equiv}$. However, it is often more convenient (in particular, when undertaking mechanised proofs) to work directly with the initial preordered set rather than with the quotient set.

We expect that the more precise the input of the semantic function, the more precise its output. We should thus consider only monotonic semantic functions. Then, function F should also be monotonic, for a suitable order on semantic functions: given more precise subcomputation results, the function F should produce more precise computation results. Formally, we adopt the following definitions.

Monotonic map. A *monotonic map* $f : A \to B$ between two preordered sets A and B is a map that preserves the preorder, that is, for all x_1, x_2 in A, if $x_1 \leq x_2$, then $f(x_1) \leq f(x_2)$. We write $A \to^m B$ for the set of monotonic maps from A to B.

Pointwise order on functions. Given a set A and a preordered set B, one can define a canonical preorder on functions in $A \to B$ by:

$$f \leq g \quad \text{iff} \quad \forall x \in A, f(x) \leq g(x).$$

Monotonic functions $A \to^m B$ between two preordered sets A and B can be ordered canonically in the same way.

Then, we have the expected result.

FACT 2. *Function F is a monotonic self-map on functions $(X \to V) \to^m T \to V$.*

$$\dfrac{\overset{\textsc{Abs}}{\emptyset \vdash \Delta \leadsto (\Delta)_\emptyset} \qquad \overset{\textsc{Abs}}{\emptyset \vdash \Delta \leadsto (\Delta)_\emptyset} \qquad \dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (\Delta)_\emptyset} \qquad \overset{\textsc{Var}}{\rho \vdash x \leadsto (\Delta)_\emptyset} \qquad \dfrac{\vdots}{\rho \vdash x\,x \leadsto v}\,\textsc{App}}{\rho \vdash x\,x \leadsto v}\,\textsc{App}}{\emptyset \vdash \Delta\,\Delta \leadsto v}\,\textsc{App}$$

where $\rho = x \mapsto (\Delta)_\emptyset$.

Figure 5. Incorrect derivation.

$$\dfrac{\dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (\Pi)_\emptyset} \qquad \dfrac{\dfrac{\overset{\textsc{Var}}{\rho \vdash x \leadsto (\Pi)_\emptyset} \qquad \rho \vdash (x\,x, x\,x) \leadsto \texttt{diverge}}{\rho \vdash x\,x \leadsto \texttt{diverge}}\,\textsc{App}}{\rho \vdash (x\,x, x\,x) \leadsto (\texttt{diverge}, \texttt{diverge})}\,\textsc{Pair}}{\rho \vdash x\,x \leadsto (\texttt{diverge}, \texttt{diverge})}\,\textsc{App}}{\rho \vdash (x\,x, x\,x) \leadsto ((\texttt{diverge}, \texttt{diverge}), (\texttt{diverge}, \texttt{diverge}))}\,\textsc{Pair}$$

where $\rho = x \mapsto (\Pi)_\emptyset$.

Figure 6. Truncated derivation.

We are onto something. But, remember we have defined F for finite values only. The next step is to define infinite values and extend F to deal with them. We want infinite values to be limits of a set of finite values. We need to make a number of definition to precise this notion of limit. In particular, for preorders, the natural notion of limit is the *supremum*.

Least element and supremum. A *least element* x of a subset s of a preordered set A is an element of s which is smaller than all other elements in s. The *supremum* x of a subset s of a preordered set A, if it exists, is the least element of A that is greater or equal to all elements of subset s. Suprema and least elements are each unique up to equivalence.

Directed subset. A subset s of a preordered set A is a *directed subset* of A if it is nonempty and every pair of its elements has an upper bound : for all x and y in s, there exists z in s such that $x \leq z$ and $y \leq z$. Directed sets are a generalisation of chains (that is, totally ordered subsets of a preordered set). Intuitively, if a directed subset s has a limit (its supremum), then given two elements of s, it is always possible to find an element of s that provides a better approximation of the limit.

Directed complete preordered set (DCPO). A *directed complete preordered set* is a preordered set such that each of its directed subsets has a supremum. Note that $DCPO$ normally stands for directed complete (partial) *order*. We use a more general definition in the present paper.

We now define the set of possibly infinite values as a completion of the set of values V, so that infinite values $\bar{v}$ be the limits of finite values that approximate them. The idea is similar to the way real numbers can be constructed as a completion of the rational numbers. We first define the objects that we want to converge. In the case of real numbers, these are Cauchy sequences. Here, we consider directed sets of values. Then, we specify how their expected limits should be related. In other words, we define an equivalence relation between Cauchy sequences or, here, a preorder on directed set. Finally, the completed set is obtained by quotienting by the equivalence relation. It appears that we can reason directly on the preorder. We thus skip this last step.

Preorder completion. The preorder completion $\overline{A}$ of a preordered set A is the set of directed subsets of A, with the following preorder:

$$s \leq t \quad \text{iff} \quad \forall x \in s, \exists y \in t, x \leq y.$$

Intuitively, for any approximation x in s, there should exist a more precise approximation y in t. The set $\overline{A}$ is a DCPO.

A supremum of a directed set d of elements of $\overline{A}$ is their union: $\sup d = \bigcup_{s \in d} s$. The function $i : x \mapsto \{x\}$ is a monotonic injection of A into $\overline{A}$ (in the relaxed sense that, if $i(x) \equiv i(y)$, then $x \equiv y$). Thus, one can consider A as if it was a subset of $\overline{A}$.

This completion is equivalent to the standard ideal completion, where an *ideal* is a downward closed directed subset. Indeed, any directed subset s is equivalent to its downward closure $\{x \in A \mid \exists y \in s, x \leq y\}$.

We thus define the set of possibly infinite values as being $\overline{V}$, the completion of the set of finite values V. We should now extend the definition of function F to also deal with infinite values. There is a canonical way to do it and get a continuous function (that is, a morphism between DCPO).

Continuous function. A function f between two preordered sets A and B is *continuous* if it preserves suprema of directed set, that is, if for any directed subset s of A, if $x \in A$ is a supremum of s, then $f(x)$ is a supremum of $f(s)$. We write $A \to^c B$ for the set of continuous functions from A to B. A continuous function is monotonic.

A monotonic function $f : A \to^m B$ can be extended in a unique way, up to equivalence, into a continuous function $\uparrow f : \overline{A} \to^c \overline{B}$ such that, for all $x \in A$, we have $\uparrow f(\{x\}) \equiv \{f(x)\}$ (this equation states that $\uparrow f$ coincides with f for all elements of set A). This function can be defined by: $\uparrow f(X) = \{f(x) \mid x \in X\}$.

Thus, we consider the continuous function $\uparrow F$. We need to show that it has a fixpoint. We rely on the following theorem.

Kleene fixpoint theorem. A *complete preordered set (CPO)* is a *directed complete preordered set* with a least element $\bot$. Every continuous self-map f on a CPO has a least fixpoint $\operatorname{lfp} f$ (such that $f(\operatorname{lfp} f) \equiv \operatorname{lfp}$), which is a supremum of the iterates $\{\bot, f(\bot), f(f(\bot)), \ldots, f^n(\bot), \ldots\}$.

The set $\overline{(X \to V) \to^m T \to V}$ is a CPO, with least element the set $\{\lambda\rho.\lambda t.\mathtt{diverge}\}$. Hence, applying the theorem to $\uparrow F$, we get a least fixpoint, which is the semantics of the calculus. Or is it? The fixpoint $\operatorname{lfp} \uparrow F$ is in $\overline{(X \to V) \to^m T \to V}$. But we would like the semantics to be in $(X \to \overline{V}) \to^m T \to \overline{V}$! How far are we from that? As we show now, the two sets can be related in an appropriate fashion, so that $\operatorname{lfp} \uparrow F$ can indeed be considered as a semantics.

For that, we need to understand how preorder completion commutes with arrows. As we will show, the set $\overline{A \to B}$, where A is a set and B is a preordered set, can be seen as a refinement of the set $A \to \overline{B}$, in the sense that the second set is equivalent to a partition of the first set. The sets $\overline{A \to^m B}$ and $\overline{A} \to^c \overline{B}$, where A and B are both preordered sets, are related in a similar fashion. The relations between these pairs of sets will be materialised by pairs of functions. Then, by combining appropriately the fixpoint $\operatorname{lfp} \uparrow F$ with these functions, we get a semantics of the expected type. We need to introduce a number of definitions to state this precisely.

Galois connection. A *Galois connection* [13] between two preordered sets A and B is a pair of functions $f : A \to B$ and $g : B \to A$ such that for all x in A and y in B we have $f(x) \leq y$ if and only if $x \leq g(y)$. In this situation, function f is called the *lower adjoint* of function g and function g is called the *upper adjoint* of function f.

If (f, g) is a Galois connection, then function f is continuous and function g is monotonic. Besides, given function f, function g is unique up to equivalence, as $g(y)$ is a supremum of $\{x \in A \mid f(x) \leq y\}$ for all y in B.

Preorder equivalence. Two preordered sets A and B are *equivalent*, written $A \approx B$, if there exists a pair of functions $f : A \to^m B$ and $g : B \to^m A$ such that for all x in A, $g(f(x)) \equiv x$ and for all y in B, $f(g(y)) \equiv y$. In other words, two preordered sets are equivalent when their associated quotient ordered sets are isomorphic.

If two preordered sets A and B are equivalent, as witnessed by two functions $f : A \to^m B$ and $g : B \to^m A$, then (f, g) is a Galois connection. Conversely, given two preordered sets A and B and two functions $f : A \to B$ and $g : B \to A$, if both (f, g) and (g, f) are Galois connections, then A and B are equivalent.

Reflection. A *reflection* [13, 23] between two preordered sets A and B is a Galois connection (f, g) between A and B such that for all y in B, $f(g(y)) \equiv y$.

The existence of a reflection between two preordered sets A and B makes it possible to consider set A as a refinement of set B, in the sense that B is equivalent to a partition of A. More precisely, a Galois connection (f, g) between two preordered sets A and B induces an equivalence relation $\sim$ on A compatible with the preorder: $x_1 \sim x_2$ iff $f(x_1) \equiv f(x_2)$. Then, the Galois connection (f, g) is a reflection when the preordered sets $A/\sim$ and B are equivalent.

Given a set A and a preordered set B, we thus want to define a reflection (G^*, G_*) between sets $\overline{A \to B}$ to $A \to \overline{B}$. Though these two functions depend on sets A and B, we do not index them, as the involved sets will always be clear from the context. The function G^* should basically be the identity for functions in $A \to B$. By using the appropriate injection functions, a function $f \in A \to B$ corresponds to $\{f\}$ in $\overline{A \to B}$ and $\lambda x.\{f(x)\}$ in $A \to \overline{B}$. We should therefore have

$$G^*(\{f\})(x) = \{f(x)\}.$$

Function G^* should also be continuous for (G^*, G_*) to be a Galois connection. Thus, we should have, for all $s \in \overline{A \to B}$ and all $x \in A$,

$$\begin{array}{rcl} G^*(s)(x) & \equiv & G^*(\sup\{\{f\} \mid f \in s\})(x) \\ & \equiv & (\sup\{G^*(\{f\}) \mid f \in s\})(x) \\ & \equiv & \sup\{G^*(\{f\})(x) \mid f \in s\} \\ & \equiv & \sup\{\{f(x)\} \mid f \in s\} \\ & \equiv & \{f(x) \mid f \in s\}. \end{array}$$

We thus take

$$G^*(s)(x) = \{f(x) \mid f \in s\}.$$

Given the characterisation of G^* in function of G_* for Galois connections, one can show that one must have

$$G_*(f) \equiv \{g \in A \to B \,|\, \forall x, \{g(x)\} \leq f(x)\}.$$

This can be taken as a definition of $G_*(f)$ by replacing the equivalence by an equality. We have the expected result.

THEOREM 3. *The pair (G^*, G_*) is a reflection.*

It is instructive to understand why we do not have a preorder equivalence but only a reflection. For that, consider the sequence of functions f_i from natural numbers to values defined by

$$f_i(j) = \begin{cases} \texttt{error} & \text{if } i \leq j \\ \texttt{diverge} & \text{otherwise} \end{cases}$$

and the function f defined by $f(j) = \texttt{error}$ for all $j \in \mathbb{N}$. Then, we have $\sup\{f_i \,|\, i \in \mathbb{N}\} \equiv f$ in $\mathbb{N} \to V$, but $\sup\{\{f_i\} \,|\, i \in \mathbb{N}\} \equiv \{f_i \,|\, i \in \mathbb{N}\} \not\equiv \{f\}$ in $\overline{\mathbb{N} \to V}$. What happens is that existing suprema, when they are not also greatest elements, are not preserved by completion; instead, new suprema are added just below. Note that, however, we have $G_*(G^*(\{f_i \,|\, i \in \mathbb{N}\})) \equiv \{f\}$, and thus, $G^*(\{f_i \,|\, i \in \mathbb{N}\}) \equiv G^*(\{f\})$: the sets $\{f_i \,|\, i \in \mathbb{N}\}$ and $\{f\}$ cannot be distinguished when considered as functions in $\mathbb{N} \to \overline{V}$. This kind of situation does indeed happen with our semantic function, with any sequence of terms that take longer and longer to yield a result. This example also suggests that one could get an equivalence by restricting appropriately the sets of functions considered so that it contains no problematic sequence of functions such as functions f_i above. This idea is developed in Section 3.2.

We similarly relate the sets $\overline{A \to^m B}$ and $\overline{A} \to^c \overline{B}$ by the pair of functions (H^*, H_*) defined by

$$\begin{gathered} H^*(\overline{f})(\overline{x}) = \{f(x) \,|\, f \in \overline{f}, x \in \overline{x}\} \\ H_*(g) = \{f \in A \to^m B \,|\, \forall x, \{f(x)\} \leq g(\{x\})\}. \end{gathered}$$

The two functions are well-defined and suitably related.

THEOREM 4. *The pair (H^*, H_*) is a reflection.*

We can finally define the semantics of a term $t \in T$ in an environment $\widehat{\rho} \in X \to \overline{V}$, as a value in $\overline{V}$:

$$[\![t]\!]_{\widehat{\rho}} = (G^* \circ H^*(\mathrm{lfp}\uparrow F) \circ G_*)(\widehat{\rho})(t).$$

There is a slight rough point with this definition. Indeed, while functions G^* and H^* are both continuous (as lower adjoints of a Galois connection), function G_* is only monotonic. When computing $[\![t]\!]_{\widehat{\rho}}$, we therefore need to approximate $G_*(\widehat{\rho}) \in \overline{X \to V}$, not directly $\widehat{\rho} \in X \to \overline{V}$. In other words, a kind of uniform convergence is required: we need to provide a directed set of functions in $X \to V$ as approximations, rather than providing for each variable x in X a directed set of values. For these reasons, we find it useful to also state the semantics of a term $t \in T$ in an environment $\overline{\rho} \in \overline{X \to V}$:

$$[\![t]\!]_{\overline{\rho}} = (G^* \circ H^*(\mathrm{lfp}\uparrow F))(\overline{\rho})(t).$$

To provide better insight on the semantics, it is interesting to unravel the definitions. We consider an environment $\overline{\rho} \in \overline{X \to V}$ and a term $t \in T$. We compute $(G^* \circ H^*(\mathrm{lfp}\uparrow F))(\overline{\rho})(t) \in \overline{V}$. Function F is a self-map on the preordered set $\Sigma = (X \to V) \to^m T \to V$. This set has a least element $\bot_\Sigma = \lambda\rho.\lambda t.\texttt{diverge}$. Then, function $\uparrow F$ is a self-map on the CPO $\overline{\Sigma}$ and $\bot_{\overline{\Sigma}} = \{\bot_\Sigma\}$ is a least element of this CPO. The semantic function is defined using Kleene fixpoint theorem as the following least fixpoint of $\uparrow F$:

$$\mathrm{lfp}\uparrow F = \sup\{(\uparrow F)^n(\bot_{\overline{\Sigma}}) \,|\, n \in \mathbb{N}\}.$$

We have, by definition, $(\uparrow F)(X) = \{F(x) \,|\, x \in X\}$. Hence, $(\uparrow F)^n(\bot_{\overline{\Sigma}}) = \{\{F^n(\bot_\Sigma)\}\}$. Then, by definition of sup for a completed preorder,

$$\begin{aligned} \mathrm{lfp}\uparrow F &= \sup\{\{F^n(\bot_\Sigma)\} \,|\, n \in \mathbb{N}\} \\ &= \{F^n(\bot_\Sigma) \,|\, n \in \mathbb{N}\}. \end{aligned}$$

Then, we have, by definition of H^*,

$$\begin{aligned} H^*(\mathrm{lfp}\uparrow F)(\overline{\rho}) &= H^*(\{F^n(\bot_\Sigma) \,|\, n \in \mathbb{N}\})(\overline{\rho}) \\ &= \{F^n(\bot_\Sigma)(\rho) \,|\, n \in \mathbb{N}, \rho \in \overline{\rho}\}. \end{aligned}$$

Finally, by definition of G^*,

$$\begin{aligned} [\![t]\!]_{\overline{\rho}} &= (G^* \circ H^*(\mathrm{lfp}\uparrow F))(\overline{\rho})(t) \\ &= G^*(\{F^n(\bot_\Sigma)(\rho) \,|\, n \in \mathbb{N}, \rho \in \overline{\rho}\})(t) \\ &= \{F^n(\bot_\Sigma)(\rho)(t) \,|\, n \in \mathbb{N}, \rho \in \overline{\rho}\}. \end{aligned}$$

The expression $F^n(\bot_\Sigma)$ is the set of assertions $\rho \vdash t \rightsquigarrow v$ that can be deduced by building approximate derivations of height n. So, the value of term t in environment $\overline{\rho}$ is the limit of the values derived from more and more precise derivation trees, and for better and better approximations ρ of environment $\overline{\rho}$. This is what we expected.

To summarize this section, we define the semantics through inductive rules only involving finite values. We prove that these rules define a monotonic operator F that maps a preordered set of functions onto itself (Facts 1 and 2). These are the only proofs that have to be adapted for other calculi. They indicate, in particular, that there is no missing nor overlapping rules. Such an operator can automatically be extended to work on the CPO obtained by completion of the preordered set of functions. There, it has a least fixpoint, which we consider as the semantics of the calculus. Though this fixpoint is not a function mapping environments and terms to values, it can be considered as such, in a canonical way, by a suitable use of reflections.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma; x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x.t : \tau_1 \to \tau_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \tau_1 \to \tau_2 \\ \Gamma \vdash t_2 : \tau_1\end{array}}{\Gamma \vdash t_1\, t_2 : \tau_2} \qquad \frac{\begin{array}{c}\Gamma \vdash t_1 : \tau_1 \\ \Gamma \vdash t_2 : \tau_2\end{array}}{\Gamma \vdash (t_1, t_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \texttt{fst}\, t : \tau_1} \qquad \frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \texttt{snd}\, t : \tau_2}$$

Figure 8. Typing rules for terms.

Values

$$\frac{\rho : \Gamma \qquad \Gamma; x : \tau_1 \vdash t : \tau_2}{(\lambda x.t)_\rho : \tau_1 \tau_2} \qquad \frac{v_1 : \tau_1 \qquad v_2 : \tau_2}{(v_1, v_2) : \tau_1 \times \tau_2}$$

$$\texttt{diverge} : \tau$$

Environments

$$\rho : \emptyset \qquad \frac{\rho : \Gamma \qquad \rho(x) : \tau}{\rho : \Gamma; x : \tau}$$

Figure 9. Typing rules for values.

2.3 Type Soundness

We illustrate how to reason on the semantics by proving a type soundness result. We use simple infinite types: types are defined coinductively by the grammar below.

$$\begin{array}{lcll} \tau & ::= & \tau \to \tau & \text{function type} \\ & & \tau \times \tau & \text{pair type} \end{array}$$

The use of infinite types ensures that interesting programs can be typed while having a very short grammar of types and few typing rules. In particular, the fixpoint operator

$$Y = \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))$$

is well typed, with type $(\tau \to \tau) \to \tau$ for any type τ.

The typing relation $\Gamma \vdash t : \tau$, where environments Γ are sequence of bindings $x : \tau$, is defined using inductive rules in Figure 8. These typing rules are standard.

In order to prove type soundness, we define a typing relation for values. We first define inductively the type of finite values (Figure 9). We want to extend this definition to all values by continuity. Given a type τ, the predicate $v : \tau$ can be seen as a function from finite values V to Booleans.

The CPO of Booleans. The set $\texttt{Bool} = \{\texttt{true}, \texttt{false}\}$ of *Booleans*, ordered such that $\texttt{true} \leq \texttt{false}$, is a CPO. Besides, this set $\texttt{Bool}$ and its completion $\overline{\texttt{Bool}}$ are equivalent: the function that maps Boolean b to the set $\{b\}$ and the function that maps set $s \in \overline{\texttt{Bool}}$ to its greatest element (which always exists) are both monotonic and inverse of one-another.

A *predicate* on a set A is a function in $A \to \texttt{Bool}$. Given a preordered set A, a monotonic predicate P on A, that is a function in $A \to^m \texttt{Bool}$, can be extended into a continuous predicate $\widetilde{P} = \sup \circ (\uparrow P)$ on $\overline{A}$. By unfolding definitions, one can see that $\widetilde{P}(s)$ holds when $P(x)$ holds for all x in s.

Hence, if typing rules $v : \tau$ are monotonic with respect to values v, we can extend them to infinite values.

FACT 5. *The typing rules for values define, for each type, a monotonic predicate on finite values. The same holds for the typing rules for environments.*

We can therefore adopt the following definitions to type values $\overline{v} \in \overline{V}$ and environments $\overline{\rho} \in \overline{X \to V}$:

- $\overline{v} : \tau$ when, for all $v \in \overline{v}$, we have $v : \tau$;
- $\overline{\rho} : \Gamma$ when, for all $\rho \in \overline{\rho}$, we have $\rho : \Gamma$.

The soundness theorem can now be stated as follows.

THEOREM 6 (Type Soundness). *If $\Gamma \vdash t : \tau$ then, for all $\overline{\rho} \in \overline{X \to V}$ such that $\overline{\rho} : \Gamma$, we have $[\![t]\!]_{\overline{\rho}} : \tau$.*

The result can also be stated for environments $\widehat{\rho} \in X \to \overline{V}$. This is an immediate consequence of the theorem.

COROLLARY 7. *If $\Gamma \vdash t : \tau$ then, for all $\widehat{\rho} \in X \to \overline{V}$ such that $\widehat{\rho} : \Gamma$ (that is, $G_*(\widehat{\rho}) : \Gamma$), we have $[\![t]\!]_{\widehat{\rho}} : \tau$.*

To prove the theorem, we need a way to reason about the semantics. The characterisation of the least fixpoint given by Kleene theorem provides such a reasoning principle.

Induction principles. The standard reasoning principle can be stated as follows. An *admissible predicate* is a continuous map from a CPO A to the CPO of Booleans. Let f be a continuous self-map on a CPO A. Suppose that an admissible predicate P on A is such that:

- $P(\bot)$;
- for all x in A, we have $P(x)$ implies $P(f(x))$.

Then $P(\operatorname{lfp} f)$ holds.

When the CPO is the completion $\overline{A}$ of a preordered set A, a variant of this induction principle allows to reason only on A rather than on $\overline{A}$. Let f be a monotonic self-map on a preordered set A with a least element $\bot$. Suppose that a monotonic predicate P on A is such that:

- $P(\bot)$;
- for all x in A, we have $P(x)$ implies $P(f(x))$.

Then $\widetilde{P}(\operatorname{lfp} \uparrow f)$ holds.

This variant relies on equality $(\uparrow f)^n(\{\bot\}) = \{f^n(\bot)\}$, from which one can deduce $\operatorname{lfp} \uparrow f = \{f^n(\bot) \mid n \in \mathbb{N}\}$.

Core rules

$$\textsc{Var}\quad \rho \vdash x \rightsquigarrow \rho(x) \qquad \textsc{Abs}\quad \rho \vdash \lambda x.t \rightsquigarrow (\lambda x.t)_{\rho} \qquad \textsc{Pair}\quad \frac{\rho \vdash t_1 \rightsquigarrow v_1 \qquad \rho \vdash t_2 \rightsquigarrow v_2}{\rho \vdash (t_1, t_2) \rightsquigarrow (v_1, v_2)}$$

$$\textsc{App}\quad \frac{\rho \vdash t \rightsquigarrow (\lambda x.t'')_{\rho'} \qquad \rho \vdash t' \rightsquigarrow v' \qquad \rho'; x \mapsto v' \vdash t'' \rightsquigarrow s}{\rho \vdash t\, t' \rightsquigarrow s} \qquad \textsc{Fst}\quad \frac{\rho \vdash t \rightsquigarrow (v_1, v_2)}{\rho \vdash \texttt{fst}\, t \rightsquigarrow v_1} \qquad \textsc{Snd}\quad \frac{\rho \vdash t \rightsquigarrow (v_1, v_2)}{\rho \vdash \texttt{snd}\, t \rightsquigarrow v_1}$$

Divergence

$$\textsc{Pair-Strict-L}\quad \frac{\rho \vdash t_1 \rightsquigarrow \texttt{diverge}}{\rho \vdash (t_1, t_2) \rightsquigarrow \texttt{diverge}} \qquad \textsc{Pair-Strict-R}\quad \frac{\rho \vdash t_1 \rightsquigarrow v_1 \qquad \rho \vdash t_2 \rightsquigarrow \texttt{diverge}}{\rho \vdash (t_1, t_2) \rightsquigarrow \texttt{diverge}} \qquad \textsc{Fst-Strict}\quad \frac{\rho \vdash t \rightsquigarrow \texttt{diverge}}{\rho \vdash \texttt{fst}\, t \rightsquigarrow \texttt{diverge}}$$

$$\textsc{App-Strict-L}\quad \frac{\rho \vdash t \rightsquigarrow \texttt{diverge}}{\rho \vdash t\, t' \rightsquigarrow \texttt{diverge}} \qquad \textsc{App-Strict-R}\quad \frac{\rho \vdash t \rightsquigarrow (\lambda x.t'')_{\rho'} \qquad \rho \vdash t' \rightsquigarrow \texttt{diverge}}{\rho \vdash t\, t' \rightsquigarrow \texttt{diverge}} \qquad \textsc{Snd-Strict}\quad \frac{\rho \vdash t \rightsquigarrow \texttt{diverge}}{\rho \vdash \texttt{snd}\, t \rightsquigarrow \texttt{diverge}}$$

Errors

$$\textsc{Pair-Error-L}\quad \frac{\rho \vdash t_1 \rightsquigarrow \texttt{error}}{\rho \vdash (t_1, t_2) \rightsquigarrow \texttt{error}} \qquad \textsc{Pair-Error-R}\quad \frac{\rho \vdash t_1 \rightsquigarrow v_1 \qquad \rho \vdash t_2 \rightsquigarrow \texttt{error}}{\rho \vdash (t_1, t_2) \rightsquigarrow \texttt{error}} \qquad \textsc{Fst-Error}\quad \frac{\rho \vdash t \rightsquigarrow s \qquad s \not\lesssim (v_1, v_2)}{\rho \vdash \texttt{fst}\, t \rightsquigarrow \texttt{error}}$$

$$\textsc{App-Error-L}\quad \frac{\rho \vdash t \rightsquigarrow s \qquad s \not\lesssim (\lambda x.t'')_{\rho'}}{\rho \vdash t\, t' \rightsquigarrow \texttt{error}} \qquad \textsc{App-Error-R}\quad \frac{\rho \vdash t \rightsquigarrow (\lambda x.t'')_{\rho'} \qquad \rho \vdash t' \rightsquigarrow \texttt{error}}{\rho \vdash t\, t' \rightsquigarrow \texttt{error}} \qquad \textsc{Snd-Error}\quad \frac{\rho \vdash t \rightsquigarrow s \qquad s \not\lesssim (v_1, v_2)}{\rho \vdash \texttt{snd}\, t \rightsquigarrow \texttt{error}}$$

Figure 10. Call-by-value semantics.

We use this theorem to prove soundness. We consider the following predicate $P(\sigma)$ that states that semantic relation $\sigma \in \Sigma = (X \to V) \to^m T \to V$ satisfies type soundness:

> if $\Gamma \vdash t : \tau$ then, for all environments $\rho \in X \to V$ such that $\rho : \Gamma$, we have $\sigma(\rho)(t) : \tau$.

We want to apply the induction principle on P. Hence, we check its premises.

FACT 8. *The predicate P is monotonic. Besides, we have $P(\bot_\Sigma)$, where $\bot_\Sigma = \lambda\rho.\lambda t.\texttt{diverge}$, and, for all $\sigma \in \Sigma$, if $P(\sigma)$ then $P(F(\sigma))$.*

The first two propositions are immediate. Regarding the last proposition, we show $P(F(\sigma))$ by case on the derivation of assumption $\Gamma \vdash t : \tau$, applying the hypothesis $P(\sigma)$ on immediate subderivations. All cases are easy.

Hence, we have $\widetilde{P}(\mathrm{lfp}\uparrow F)$: the semantics satisfies the soundness property. We are not quite done yet, however: the statement of Theorem 6 is not an immediate consequence of this result. Let us now conclude the proof. We assume $\Gamma \vdash t : \tau$. Let $\overline{\rho} \in \overline{X \to V}$ such that $\overline{\rho} : \Gamma$. We now need to prove that $[\![t]\!]_{\overline{\rho}} : \tau$. Note that $[\![t]\!]_{\overline{\rho}} = \{\sigma(t)(\rho) \mid \sigma \in \mathrm{lfp}\uparrow F, \rho \in \overline{\rho}\}$. Hence, it is sufficient to prove that for all $\sigma \in \mathrm{lfp}\uparrow F$ and all $\rho \in \overline{\rho}$ we have $\sigma(t)(\rho) : \tau$. Let $\sigma \in \mathrm{lfp}\uparrow F$ and $\rho \in \overline{\rho}$. We have $\overline{\rho} : \Gamma$, hence $\rho : \Gamma$. On the other hand, $\widetilde{P}(\mathrm{lfp}\uparrow F)$, hence $P(\sigma)$ holds for all $\sigma \in \mathrm{lfp}\uparrow F$. This is exactly what we need to conclude.

Soundness is thus fairly easy to establish. We do not have to work directly with possibly infinite values nor infinite derivations. Instead, the proof is based on an inductive principle, which we are able to apply without proving any admissibility result, and that involves only finite values and derivations. Only a bit of fiddling is needed at the end to relate the soundness statement resulting from this principle with a direct statement of type soundness.

3. Considering Other Calculi

We discuss how our technique can be applied to specify the semantics of other calculi.

3.1 Deterministic Call-by-Value Calculi

We define the semantics of a call-by-value calculus. This is simpler as there are only finite values. We keep the same syntax for terms (Section 2.1). In a call-by-value settings, errors and divergence cannot occur nested inside a value.

Thus, we distinguish the values $v \in V$ of the calculus from the *semantic values* $s \in S$. We have the following grammars.

$$\begin{array}{lcll} v & ::= & (\lambda x.t)_\rho & \text{function closure} \\ & & (v,v) & \text{pair} \\ s & ::= & v & \text{regular value} \\ & & \texttt{diverge} & \text{divergence} \\ & & \texttt{error} & \text{error} \end{array}$$

Semantic values are ordered as follow: $s_1 \leq s_2$ when $s_1 = \texttt{diverge}$. This is a so-called *flat CPO*. An interesting property of flat CPOs is that they are equivalent to their completion: $\overline{S} \approx S$. This is because all suprema are reached. The canonical injection function maps set S to set $\overline{S}$; the converse mapping is provided by the supremum function. Note that Booleans are another instance of flat CPOs.

The semantics is a relation $\rho \vdash t \rightsquigarrow s$ between an environment $\rho \in X \to V$, a term $t \in T$ and a semantic value $s \in S$. We define it by the inference rules in Figure 10. One can check that the operator F specified by these rules is a monotonic self-map on

$$(X \to V) \to T \to S.$$

As previously, we extend F into a self-map $\uparrow F$ on CPO:

$$\overline{(X \to V) \to T \to S}.$$

The semantics is the least fixpoint $\mathrm{lfp} \uparrow F$ of this operator. To use it, one should compose it with appropriate reflections. We thus adopt the following definition:

$$[\![t]\!]_\rho = \sup(G^*(G^*(\mathrm{lfp} \uparrow F)(\rho))(t)).$$

This gives us a function with signature

$$(X \to V) \to T \to S,$$

which is exactly the expected signature of a semantics.

Note that we could have left implicit either the rules for divergence, or the rule for errors. Then, the inference rules would define a partial operator F'. When the divergence rules are omitted, this operator can be extended into the total operator F by

$$F(\sigma)(\rho)(t) = \begin{cases} F'(\sigma)(\rho)(t) & \text{when defined} \\ \texttt{diverge} & \text{otherwise.} \end{cases}$$

One can then proceed as previously to define the semantics.

3.2 Maps with Finite Support

We have shown in Section 2.2 that the DCPOs $\overline{A \to B}$ and $A \to \overline{B}$, where A is a set and B is a preordered set, are not equivalent in general. One can show, however, that the equivalence holds when the set A is finite.

Another interesting case is when one restricts oneself to functions with finite support. Let A be a set and B be a preordered set with a distinguished *maximal element* m (that is, for all y in B, if $m \leq y$, then $m \equiv y$). We say that a function $g \in A \to B$ has *finite support* when $g(x) = m$ for all x in A but a finite number of them. We write $A \to^f B$ for the set of such functions. One can then show that $\overline{A \to^f B}$ and $A \to^f \overline{B}$ are equivalent, with witness the pair of functions (G^*, G_*). defined previously, restricted to these sets. As a consequence of the equivalence, both these functions are continuous.

It is quite usual to consider environments to be functions from variables X to values $\overline{V}$ with finite support (taking `error` as distinguished element). This solves the issue we encountered in Section 2.2 where we saw that environments had to be approximated in $\overline{X \to V}$ rather than in $X \to \overline{V}$ for lack of continuity. The price to pay is a more complex definition of environments.

3.3 Non-Determinism

As we mentioned in Section 2.2, it would be interesting to extend our running example with a non-deterministic choice. Indeed, we explained that this operator makes it possible to distinguish call-by-need from call-by-name. But then, clearly, the semantics is not going to be a function returning a single value anymore.

The usual approach to deal with non-determinism is to use powerdomains (the order-theoretic counterparts to powersets), such as Plotkin powerdomains [25]. Then, the semantic is a function returning an element in a powerdomain of values. But powerdomains are complex beasts, and thus we loose some of the simplicity of our approach.

An alternative due to Cary and Sarkar [11] is to use oracles: the semantics is made deterministic by threading an oracle, a sequence of Boolean $\Omega = (\omega_n)_{n \in \mathbb{N}}$, through the computation. The semantic relation $\rho \vdash_\Omega t \rightsquigarrow v$, can then associate a single value to a triple of an oracle, an environment and a term. Whenever a Boolean is needed (for instance, to choose an alternative for a non-deterministic choice), one can take the first element ω_0 of the sequence; it remains the oracle $(\omega'_n)_{n \in \mathbb{N}}$ such that $\omega'_n = \omega_{n+1}$. An oracle can also be split into two independent oracles $(\omega^1_n)_{n \in \mathbb{N}}$ and $(\omega^2_n)_{n \in \mathbb{N}}$ with $\omega^1_n = \omega_{2n}$ and $\omega^2_n = \omega_{2n+1}$. This is useful when several computations are performed independently. This would be the case for the evaluations of the function argument and the function body in rule App of Figure 1.

4. Mechanised Formalisation

Our results regarding the call-by-need calculus of Section 2 have been formalised in Coq [30]. Our packaging of mathematical structures (preorders, DCPOs and CPOs, and the corresponding morphisms) follows the techniques established by Garillot, Gonthier, Mahboubi and Rideau [15, 16].

We take Coq propositions `Prop` for the CPO of Booleans. We have the following hierarchy of structures:

- a preorder is a set `A` with a reflexive transitive relation `sub : A → A → Prop`;

- a DCPO is a preorder with a function `sup` of signature `directed_set A` $\to$ `A`, which associates to each directed set its supremum (a directed set `directed_set A` is a characteristic function, that is a function in `A` $\to$ `Prop`, with suitable properties);
- a CPO is a DCPO with a least element `bottom`.

The fact that `sup` is defined as a function rather than a relation makes it possible to have an explicit definition of the least fixpoint `lfp f` of a function `f`, as the function `sup` applied to the directed set containing the iterates $\mathtt{f}^{\mathtt{n}}$(`bottom`). However, due to the constructive nature of Coq, the supremum function `sup` cannot be defined for concrete CPOs such as the set S of semantic values of Section 3.1: it is not possible to associate in a constructive way a suitable element of S to the characteristic function `f` $: S \to$ `Prop` corresponding to each directed set. For the same reason, one can establish that $\overline{S}$ and S are equivalent preorders (in the sense that there exists suitable functional *relations* between the two), but it is not possible to provide an explicit function that map an element of $\overline{S}$ to an element of S.

The operator F is directly specified in Coq as a function from semantic functions to semantic functions. We use pattern matching, which helps keep the definition short by sharing some cases. Such a definition directly shows that operator F maps functions to functions (Fact 1). Then, one only needs to show that F is appropriately monotonic (Fact 2) to complete the definition of the semantics of the calculus. Establishing type soundness is similarly short and simple.

The mechanised formalisation of complex semantics was one of the main motivations for this work. We found that working with a proof assistant was also very helpful for finding the right definitions.

5. Related Work

The works by Gunter and Remy [17] and Stoughton [28] are most related to ours. They describe the incremental construction of derivation trees, which is similar to our presentation of approximate derivations. We go beyond these works by working at a more abstract level, making an explicit connection with order theory and dealing with more complex domains of values.

Leroy and Grall [20] present a trace-based big-step semantics for a call-by-value calculus. They use separately an inductive definition for converging behaviors and a coinductive definition for divergence. It would be interesting to compare this approach with ours in which both cases would be handled simultaneously. The complexity should be in-between the two calculi we have studied here: one has to deal with finite values V and environments $X \to V$ but infinite semantic values (traces) $\overline{S}$.

An alternative to our order-theoretic approach is the mixed induction and coinduction approach, advocated in particular by Danielsson and Altenkirch [12], where rules are partitioned into inductive and coinductive rules. With this approach, inference rules are separated in two sets: those interpreted in a inductive fashion, and those interpreted in a coinductive fashion. This yields two operators F_{ind} and F_{coind}. These operators are then combined into a single operator $F(X, Y) = F_{\text{ind}}(X) \cup F_{\text{coind}}(Y)$. Then, the semantics of the rules is

$$\mathrm{gfp}(\lambda Y.\, \mathrm{lfp}(\lambda X.F(X, Y))).$$

In order to reason on this semantics, one has to combine coinduction with induction. Concretely, derivation trees are allowed to be infinite, but only finitely many inductive rules may ever appear on top of one another without any coinductive rule in-between. In our case, rules for divergence should be interpreted in a coinductive way; the rule App for function application ought to be split in two, depending on whether the function body evaluates to `diverge` or not. We feel that having to combine inductive and coinductive reasoning is complex. For instance, the proof that the semantics of the calculus is deterministic would not be immediate at all with this approach: it would be a rather complex proof involving both induction and coinduction.

A similar approach, proposed by Cousot and Cousot [8, 9], is to partition instead the universe $\mathcal{U}$, relying on the following set isomorphism: $\mathcal{P}(\mathcal{U}_1 \cup \mathcal{U}_2) \approx \mathcal{P}(\mathcal{U}_1) \times \mathcal{P}(\mathcal{U}_2)$. Then, one can order $\mathcal{P}(\mathcal{U}_1)$ by inclusion and use the converse order for $\mathcal{P}(\mathcal{U}_2)$. This works well for a call-by-value semantics, where one can separate converging behaviors from diverging behaviors, but it is not clear how this framework can be extended to cover our call-by-need calculus.

We use order theory to define an operational semantics. A fruitful research topic has been to start instead from an operational semantics and build order-theoretic concepts upon this semantics. This includes defining types as sets of terms that are downward-closed and closed under limits, as in MacQueen, Plotking and Sethi [21], Abadi, Pierce and Plotkin [1], Birkedal and Harper [6], or Vouillon and Melliès [31]. It is also possible to define syntactic logical relations as in Crary and Harper [10]. As shown by Mason, Smith and Talcott [22, 27], many notions from domain theory can be defined in this context.

There have been several formalisations of domain theory in Coq. Bertot and Komendantsky [5] use a choice axiom to prove Kleen fixpoint theorem. They are able to extract possibly non-deterministic programs. During extraction, the least fixpoint operator is replaced by a fixpoint function of the target language. Paulin-Mohring [24] develop a constructive formalisation, taking advantage of the possibility to define coinductive datastructures in Coq. Benton, Birkedal, Kennedy and Varming [3, 4] extend this work up to and including the inverse-limit construction of solutions to mixed-variance recursive domain equations, and the existence of invariant relations on those solutions. Our work is different from these works as we use syntactic values as a domain: for instance, the semantics of an abstraction is syntactically a closure, not a continuous function from values to values. We have a non-constructive definition of CPOs. But it should be possible to specify our semantics in a constructive way, as they are defined as the supremum of a chain.

6. Conclusion

We have presented a way to specify big-step semantics that account simultaneously for finite and infinite behaviors, based on order theory. We were able to specify semantics with little proof obligations. Basically, one only has to prove that the inference rules define a monotonic self-map on a suitable set of functions, which is easy. It is noteworthy that we do not have to establish any continuity property (as is often the case for order-theoretic developments); indeed, this is typically a lot more complicated. Also, we only work with very concrete objects defined using inductive grammars and with finite derivations involving these objects; in particular, it is possible to reason inductively on these objects and derivations, as well as on the semantics itself. A bit more work is required than with the standard way of defining big-step semantics, but we gain a lot in precision.

References

[1] Abadi, M., Pierce, B., Plotkin, G.: Faithful ideal models for recursive polymorphic types. International Journal of Foundations of Computer Science 2(1) (Mar 1991)

[2] Aczel, P.: An introduction to inductive definitions. In: Barwise, K.J. (ed.) Handbook of Mathematical Logic. North-Holland, Amsterdam (1977)

[3] Benton, N., Birkedal, L., Kennedy, A., Varming, C.: Formalizing domains, ultrametric spaces and semantics of programming languages (2010), submitted for publication

[4] Benton, N., Kennedy, A., Varming, C.: Some domain theory and denotational semantics in Coq. In: TPHOLs '09. Springer-Verlag (2009)

[5] Bertot, Y., Komendantsky, V.: Fixed point semantics and partial recursion in Coq. In: PPDP '08. ACM (2008)

[6] Birkedal, L., Harper, R.: Constructing interpretations of recursives types in an operational setting. Information and Computation 155 (1999)

[7] Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science 25(2) (1983)

[8] Cousot, P., Cousot, R.: Inductive definitions, semantics and abstract interpretations. In: POPL '92. ACM (1992)

[9] Cousot, P., Cousot, R.: Bi-inductive structural semantics. Inf. Comput. 207 (2009)

[10] Crary, K., Harper, R.: Syntactic logical relations for polymorphic and recursive types. Electron. Notes Theor. Comput. Sci. 172 (April 2007)

[11] Crary, K., Sarkar, S.: Foundational certified code in the twelf metalogical framework. ACM Transactions on Computatational Logic 9(3) (2008)

[12] Danielsson, N.A., Altenkirch, T.: Subtyping, declaratively: an exercise in mixed induction and coinduction. In: MPC'10. Springer-Verlag (2010)

[13] Erné, M., Koslowski, J., Melton, A., Strecker, G.E.: A Primer on Galois Connections. New York Academy Sciences Annals 704, 103–125 (Dec 1993)

[14] Gapeyev, V., Levin, M.Y., Pierce, B.C.: Recursive subtyping revealed: (functional pearl). SIGPLAN Not. 35 (September 2000)

[15] Garillot, F., Gonthier, G., Mahboubi, A., Rideau, L.: Packaging mathematical structures. In: TPHOLs '09. Springer-Verlag (2009)

[16] Gonthier, G.: Type design patterns for computer mathematics. In: TLDI '11. ACM (2011)

[17] Gunter, C.A., Rémy, D.: A proof-theoretic assessment of runtime type errors. Research Report 11261-921230-43TM, AT&T Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07974-2070 (1993)

[18] Kahn, G.: Natural semantics. In: STACS '87. Springer-Verlag, London, UK (1987)

[19] Kutzner, A., Schmidt-Schauß, M.: A non-deterministic call-by-need lambda calculus. In: ICFP '98. ACM (1998)

[20] Leroy, X., Grall, H.: Coinductive big-step operational semantics. Inf. Comput. 207(2) (2009)

[21] MacQueen, D., Plotkin, G., Sethi, R.: An ideal model for recursive polymorphic types. In: POPL '84. ACM (1984)

[22] Mason, I.A., Smith, S.F., Talcott, C.L.: From operational semantics to domain theory. Information and Computation 128(1) (1996)

[23] Melton, A., Schröder, B.S.W., Strecker, G.E.: Lagois connections - a counterpart to Galois connections. Theor. Comput. Sci. 136(1), 79–107 (1994)

[24] Paulin-Mohring, C.: From Semantics and Computer Science: Essays in Honor of Gilles Kahn, chap. A constructive denotational semantics for Kahn networks in Coq. Cambridge University Press (2009)

[25] Plotkin, G.D.: A powerdomain construction. SIAM J. Comput. 5(3) (1976)

[26] Plotkin, G.D.: A structural approach to operational semantics. Tech. Rep. DAIMI FN–19, Computer Science Department, Aarhus University, Aarhus, Denmark (Sep 1981)

[27] Smith, S.F.: The coverage of operational semantics. In: Gordon, A.D., Pitts, A.M. (eds.) Higher Order Operational Techniques in Semantics. Publications of the Newton Institute, Cambridge University Press (1998)

[28] Stoughton, A.: An operational semantics framework supporting the incremental construction of derivation trees. In: Second Workshop on Higher-Order Operational Techniques in Semantics. Elsevier Science Publishers B. V. (1998)

[29] Tarski, A.: A lattice theoretical fixpoint theorem and its applications. Pacific J. of Mathematics 5 (1955)

[30] The Coq Development Team: The Coq Proof Assistant Reference Manual – Version V8.2 (2008), `http://coq.inria.fr`, `http://coq.inria.fr`

[31] Vouillon, J., Melliès, P.A.: Semantic types: a fresh look at the ideal model for types. In: POPL '04. ACM (2004)