Semantic Types: A Fresh Look at the Ideal Model for Types

Jérôme Vouillon

Paul-André Melliès

CNRS and Denis Diderot University
{Jerome.Vouillon,Paul-Andre.Mellies}@pps.jussieu.fr

Abstract

We present a generalization of the ideal model for recursive polymorphic types. Types are defined as sets of terms instead of sets of elements of a semantic domain. Our proof of the existence of types (computed by fixpoint of a typing operator) does not rely on metric properties, but on the fact that the identity is the limit of a sequence of projection terms. This establishes a connection with the work of Pitts on relational properties of domains. This also suggests that ideals are better understood as closed sets of terms defined by orthogonality with respect to a set of contexts.

Categories and Subject Descriptors

F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Type structure*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Operational semantics*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Semantics*; D.3.2 [**Programming Languages**]: Language Classifications—*Applicative (functional) languages*

General Terms

Languages, Theory

Keywords

Recursive types, polymorphism, subtyping, realizability, ideal model, inductive/coinductive principle

1 Introduction

The typing and subtyping rules for powerful type systems can be rather complex and unexpected, due to interactions between the several kinds of types. The first author experienced this with XDuce [26], which combines union types and product types. A clear semantics of subtyping appears to be really helpful in those situations [15, 21, 27]. Ideally, it should be set-theoretic. In the case of XDuce, subtyping is simply defined as the inclusion of types as sets of values. This is quite handy: union types and product types are then directly interpreted as set-theoretic union and product.

Unfortunately, the semantic approach becomes complicated, but hopefully still intuitive, in presence of recursive types. Indeed, in this case, the denotation of a type cannot be defined by induction on its structure. Instead, it is generally defined as a fixpoint of some operator. The existence of such a fixpoint is not immediate: the operator is usually not monotone, since function types are contravariant on the left and covariant on the right.

MacQueen, Plotkin, and Sethi [31] proved the existence of such a fixpoint when the denotations of types is taken to be *ideals*, that is, sets of elements of a semantic domain enjoying additional properties. This is fine, but not entirely satisfactory. First, domain theory is complex, especially if one wants to account for non-determinism. Second, there is often a mismatch between the operational semantics of a language and its denotational semantics, which can have an impact on the properties of types. It seems therefore important to recast this model inside operational semantics. Some major steps have been taken in this direction [1, 5, 12, 13, 14]. We carry on, and provide here an alternative perspective, halfway between the work of Krivine on realizability [16] and the work of Pitts on logical relations [34, 35].

Our goal is to classify *terms* according to their *types*. We expect two properties of types: first, if a term has a type, then it evaluates without error; second, it must be possible to build more complex typed terms by assembling smaller typed terms according to simple (typing) rules. For instance:

$$\textsc{App}\ \frac{e : \tau' \to \tau \qquad e' : \tau'}{e\,e' : \tau} \qquad \textsc{Fst}\ \frac{e : \tau \times \tau'}{\mathtt{fst}\,e : \tau} \qquad \textsc{Snd}\ \frac{e : \tau \times \tau'}{\mathtt{snd}\,e : \tau'}$$

Let $\Delta(\tau)$ be the *denotation* of type τ, that is the set of terms of type τ. Let $\mathcal{S}$ be the set of terms that evaluates without error. The rules above suggest to have the following inclusions.

$$\begin{array}{lcl} \Delta(\tau' \to \tau) & \subseteq & \{e \in \mathcal{S} \mid \forall e' \in \Delta(\tau').e\,e' \in \Delta(\tau)\} \\ \Delta(\tau \times \tau') & \subseteq & \{e \in \mathcal{S} \mid \mathtt{fst}\,e \in \Delta(\tau) \wedge \mathtt{snd}\,e \in \Delta(\tau')\} \end{array}$$

POPL'04, January 14–16, 2004, Venezia, Italia.

These inclusions ensure the *soundness* of the typing rules above. In order to reason about types, it is important to have a more precise characterization of types. It seems therefore natural to replace these inclusions by an equality. We can then view Δ as a *fixpoint* of an operator F such that:

$$\begin{array}{lcl} F(\Delta)(\tau' \to \tau) & = & \{e \in \mathcal{S} \mid \forall e' \in \Delta(\tau').e\,e' \in \Delta(\tau)\} \\ F(\Delta)(\tau \times \tau') & = & \{e \in \mathcal{S} \mid \texttt{fst}\,e \in \Delta(\tau) \wedge \texttt{snd}\,e \in \Delta(\tau')\} \end{array}$$

This operator is neither monotone nor anti-monotone, as can be seen on the first equation: the set $F(\Delta)(\tau' \to \tau)$ gets larger when the set $\Delta(\tau')$ gets smaller and when the set $\Delta(\tau)$ gets larger. As we shall see, it turns out to be useful refine the picture, and to define an operator F with two arguments, monotone on its first argument and anti-monotone on its second argument:

$$\begin{array}{l} F(\Delta^+,\Delta^-)(\tau' \to \tau) = \{e \in \mathcal{S} \mid \forall e' \in \Delta^-(\tau').e\,e' \in \Delta^+(\tau)\} \\ F(\Delta^+,\Delta^-)(\tau \times \tau') = \{e \in \mathcal{S} \mid \texttt{fst}\,e \in \Delta^+(\tau) \wedge \texttt{snd}\,e \in \Delta^+(\tau')\} \end{array}$$

Still, the existence of Δ such that $F(\Delta,\Delta) = \Delta$ is not clear as the Knaster-Tarski fixpoint theorem does not apply.

This article is organized as follows. We recall some lattice theory (section 2) and provide general conditions implying the existence of a fixpoint of the typing operator F above (section 3). Then, we specify a type system and its denotation (section 4), from which we deduce the soundness of our subtyping system (section 5). The closure operator involved in these constructions is nicely expressed by orthogonality between terms and contexts (section 6). We illustrate our theory by a call-by-value calculus (section 7) and its typing rules (section 8). We finish by mentioning limitations of our approach (section 9), presenting related work (section 10), and pointing out future directions (section 11).

2 Mathematical Background

Complete Lattice A *complete lattice* is a partially ordered set such that any subset X of elements has a least upper bound $\sup X$ and a greatest lower bound $\inf X$.

Limit We define the *limit inferior* and the *limit superior* of a sequence $(x_n)_{n\in\mathbb{N}}$ of elements of a complete lattice, by:

$$\begin{array}{lcl} \liminf x_n & = & \sup\limits_{n\in\mathbb{N}} \inf\limits_{n'\geq n} x_{n'} \\ \limsup x_n & = & \inf\limits_{n\in\mathbb{N}} \sup\limits_{n'\geq n} x_{n'} \end{array}$$

When the two limits coincide, we say that the sequence *converges* and we define the *limit* of the sequence by:

$$\lim x_n = \liminf x_n = \limsup x_n$$

All increasing and decreasing sequences converge. The limit operator is order preserving: if $x_n \preceq y_n$ for all $n \in \mathbb{N}$, then $\lim x_n \preceq \lim y_n$. The *Sandwich Theorem* holds: if a sequence $(z_n)_{n\in\mathbb{N}}$ is such that $x_n \preceq z_n \preceq y_n$ for all $n \in \mathbb{N}$, where $(x_n)_{n\in\mathbb{N}}$ and $(y_n)_{n\in\mathbb{N}}$ are two sequences converging to a same limit, then $(z_n)_{n\in\mathbb{N}}$ converges to this very limit.

Closure A *closure operator* over a partially ordered set $(X,\preceq)$ is a function which associates to each element x of X an element $\overline{x}$ of X such that:

- $x \preceq \overline{x}$ (extensive);
- $\overline{\overline{x}} = \overline{x}$ (idempotent);
- if $x \preceq x'$, then $\overline{x} \preceq \overline{x'}$ (monotone).

An element x is *closed* when $\overline{x} = x$. The greatest lower bound of a family of closed elements is closed. The closed elements of a complete lattice form themselves a complete lattice.

Galois Connection Let $(X,\preceq)$ and $(Y,\sqsubseteq)$ be two partially ordered sets. A *Galois connection* is a pair of two functions $f : X \to Y$ and $g : Y \to X$ such that for all $x \in X$ and $y \in Y$, $x \preceq g(y)$ iff $f(x) \sqsubseteq y$. A Galois connection (f,g) induces a closure operator $g \circ f$ over the partially ordered set $(X,\preceq)$.

Polarity A binary relation R between two sets A and B induces a Galois connection, called *polarity*, between the two power sets $(\mathcal{P}(A),\subseteq)$ and $(\mathcal{P}(B),\supseteq)$. The components (f,g) of this Galois connection are defined by:

$$\begin{array}{ll} f(U) = \{y \in B \mid \forall x \in U.(x,y) \in R\} & \text{for } U \subseteq A \\ g(V) = \{x \in A \mid \forall y \in V.(x,y) \in R\} & \text{for } V \subseteq B \end{array}$$

3 Fixpoints: Existence and Properties

In this section, we specify some conditions ensuring the definition of the denotation $\Delta_\infty(\sigma)$ of a family of types σ as a fixpoint of an operator F. We also state some interesting properties of this fixpoint.

3.1 Assumptions on the Calculus

The calculus is composed of a set of *terms* e. Before section 7, we don't actually assume that these are syntactic terms. They could just as well be, for instance, the elements of a semantic domain.

A closure operator associates to a set of terms $\mathcal{E}$ another set of terms $\overline{\mathcal{E}}$. The intuition is that $\overline{\mathcal{E}}$ is the set of terms that cannot be distinguished from the terms $\mathcal{E}$: a term in $\overline{\mathcal{E}}$ will evaluate without error in any context where all terms in $\mathcal{E}$ would evaluate without error. A possible definition for the closure operator is proposed in section 6.

We distinguish a set $\mathcal{S}$ of *safe terms* (which, intuitively, evaluate without error) and a set $\mathcal{B}$ of *bottom terms* (which loop). These sets satisfy the following properties:

- $\mathcal{S}$ is closed;
- $\mathcal{B}$ is closed;
- $\mathcal{B} \subseteq \mathcal{S}$.

There exist a family $(\_)^n$ of *approximation operators* ($n \in \mathbb{N}$), from terms to terms, with the following properties:

- if $e \in \overline{\mathcal{E}}$, then $(e)^n \in \overline{\mathcal{E}}$;
- if $(e)^n \in \overline{\mathcal{E}}$ for all $n \in \mathbb{N}$, then $e \in \overline{\mathcal{E}}$;
- if $e \in \mathcal{S}$, then $(e)^0 \in \mathcal{B}$.

An intuition is that $(e)^n$ behaves just like e until a nested interaction of depth n is reached, then loops; and that a term e is the "limit" of its approximations $(e)^n$.

In section 7, we will fully specify a calculus and provide a corresponding definition for the sets $\mathcal{S}$ and $\mathcal{B}$, and for the approximations operators $(\_)^n$.

3.2 Semantic Candidates

We now define more precisely the domain of the operator F. We assume given a set of *types* σ. The operator F takes as arguments two semantic candidates and returns a semantic candidate. A *semantic candidate* Δ is a function from types to sets of terms which satisfies the following property:

$$\mathcal{B} \subseteq \Delta(\sigma) \subseteq \mathcal{S}.$$

In other words, a well-typed term does not fail, and a term that loops can be given any type.

The semantic candidates are naturally equipped with a complete lattice structure induced by the point-wise order $\sqsubseteq$:

$$\Delta \sqsubseteq \Delta' \text{ iff } \forall\sigma.\Delta(\sigma) \subseteq \Delta'(\sigma).$$

The closure operator on set of types induces a closure operator on semantic candidates, defined by:

$$\overline{\Delta}(\sigma) = \overline{\Delta(\sigma)}.$$

3.3 Assumptions on the Operator F

We now state some properties of the operator F ensuring that it has a fixpoint.

We define a family of relations $\lhd^n$ by:

$$\mathcal{E} \lhd^n \mathcal{E}' \text{ iff for all } e \text{ in } \mathcal{E} \text{ we have } (e)^n \in \mathcal{E}'.$$

In other words, $\mathcal{E} \lhd^n \mathcal{E}'$ iff the image of $\mathcal{E}$ by $(\_)^n$ is included in $\mathcal{E}'$. We write $\Delta \lhd^n \Delta'$ for the point-wise extension of the relation $\lhd^n$ to semantic candidates. We state a few interesting properties of the relations $\lhd^n$ that we will use in section 3.4.

Remark 1 (Some properties of the relations $\lhd^n$). *The following properties hold:*

- $\mathcal{E} \lhd^0 \mathcal{B}$ *if* $\mathcal{E} \subseteq \mathcal{S}$*;*
- *if* $\mathcal{E}_1 \subseteq \mathcal{E}'_1$, $\mathcal{E}'_1 \lhd^n \mathcal{E}'_2$, *and* $\mathcal{E}'_2 \subseteq \mathcal{E}_2$, *then* $\mathcal{E}_1 \lhd^n \mathcal{E}_2$*;*
- *if* $\mathcal{E} \lhd^n \mathcal{E}'$ *for all* $n \in \mathbb{N}$, *then* $\mathcal{E} \subseteq \overline{\mathcal{E}'}$.

Proof. The two first properties are easy to check. We prove the third one. Suppose that $e \in \mathcal{E}$. By assumption, $(e)^n \in \mathcal{E}'$ for all $n \in \mathbb{N}$. So, $e \in \overline{\mathcal{E}'}$. We conclude that $\mathcal{E} \subseteq \overline{\mathcal{E}'}$. $\square$

The operator F satisfies the three following properties:

- F is monotone on its first argument and anti-monotone on its second argument:
  if $\Delta_1 \sqsubseteq \Delta'_1$ and $\Delta'_2 \sqsubseteq \Delta_2$, then $F(\Delta_1, \Delta_2) \sqsubseteq F(\Delta'_1, \Delta'_2)$;
- F preserves closure:
  if Δ and Δ' are closed, then $F(\Delta, \Delta')$ is also closed;
- F is "contractive":
  if Δ and Δ' are closed and $\Delta \lhd^n \Delta'$, then $F(\Delta, \Delta') \lhd^{n+1} F(\Delta', \Delta)$.

The last property plays the same technical role as contractivity in [31] without requiring a metric space.

3.4 Semantic Typing

We can now state the existence of a fixpoint. Actually, it turns out that there exists a *best* fixpoint, in the sense that it is the closure of any other fixpoint, hence the greatest one. We will only consider this particular fixpoint Δ_∞, the *semantic typing*, in the remainder of this paper.

Theorem 2 (Greatest Fixpoint). *The operator F has a closed fixpoint Δ_∞. This fixpoint is the closure of any other fixpoint of F.*

Proof. **Preliminary** Let $\Delta_\mathcal{B}$ be the least semantic candidate (which associates $\mathcal{B}$ to each type) and $\Delta_\mathcal{S}$ be the greatest semantic candidate (which associates $\mathcal{S}$ to each type). We define two sequences of semantic candidates $(\Delta^-_n)_{n\in\mathbb{N}}$ and $(\Delta^+_n)_{n\in\mathbb{N}}$ by:

$$\begin{array}{ll} \Delta^-_0 = \Delta_\mathcal{B} & \Delta^-_{n+1} = F(\Delta^-_n, \Delta^+_n) \\ \Delta^+_0 = \Delta_\mathcal{S} & \Delta^+_{n+1} = F(\Delta^+_n, \Delta^-_n) \end{array}$$

The sequence $(\Delta^-_n)_{n\in\mathbb{N}}$ is increasing and the sequence $(\Delta^+_n)_{n\in\mathbb{N}}$ is decreasing. Besides, for all n, we have $\Delta^-_n \sqsubseteq \Delta^+_n$. So, both sequences are converging to some semantic candidate Δ^-_∞ and Δ^+_∞ and we have the following ordering:

$$\Delta^-_0 \sqsubseteq \ldots \sqsubseteq \Delta^-_n \sqsubseteq \ldots \sqsubseteq \Delta^-_\infty \sqsubseteq \Delta^+_\infty \sqsubseteq \ldots \sqsubseteq \Delta^+_n \sqsubseteq \ldots \sqsubseteq \Delta^+_0.$$

Furthermore, the semantic candidates Δ^-_n and Δ^+_n are closed for all $n \in \mathbb{N}$, as F preserves closure. The semantic candidate Δ^+_∞ is also closed, as it is the greatest lower bound of a set of closed semantic candidates. The semantic candidate Δ^-_∞, on the other hand, need not be closed.

Any fixpoint of F is between Δ^-_∞ and Δ^+_∞. Indeed, we can prove by induction that a fixpoint Δ satisfies $\Delta^-_n \sqsubseteq \Delta \sqsubseteq \Delta^+_n$ for all n. This suggests to compare more precisely these two semantic candidates.

A closed fixpoint candidate We are going to prove that $\Delta^+_\infty = \overline{\Delta^-_\infty}$. This implies that if there is a closed fixpoint, it must be Δ^+_∞.

First, we prove by induction that $\Delta^+_n \lhd^n \Delta^-_n$. This relation holds for $n = 0$:

$$\Delta^+_0(\sigma) \lhd^0 \mathcal{B} = \Delta^-_0(\sigma).$$

Let us assume that it holds for some n. Then, it holds for $n+1$ as F is contractive:

$$\Delta^+_{n+1} = F(\Delta^+_n, \Delta^-_n) \lhd^{n+1} F(\Delta^-_n, \Delta^+_n) = \Delta^-_{n+1}.$$

So, we have $\Delta^+_\infty \sqsubseteq \Delta^+_n$, $\Delta^+_n \lhd^n \Delta^-_n$, and $\Delta^-_n \sqsubseteq \Delta^-_\infty$. Hence, $\Delta^+_\infty \lhd^n \Delta^-_\infty$. This is true for all $n \in \mathbb{N}$, so $\Delta^+_\infty \sqsubseteq \overline{\Delta^-_\infty}$.

On the other hand, we have $\Delta^-_\infty \sqsubseteq \Delta^+_\infty$ and Δ^+_∞ is closed. Therefore, $\overline{\Delta^-_\infty} \sqsubseteq \Delta^+_\infty$.

Existence of the closed fixpoint We now check that Δ^+_∞ is indeed a fixpoint, that is, $\Delta^+_\infty = F(\Delta^+_\infty, \Delta^+_\infty)$.

We remark that for all semantic candidate Δ, if $\Delta^-_\infty \sqsubseteq \Delta \sqsubseteq \Delta^+_\infty$, then $\Delta^-_\infty \sqsubseteq F(\Delta, \Delta) \sqsubseteq \Delta^+_\infty$. Indeed, if $\Delta^-_\infty \sqsubseteq \Delta \sqsubseteq \Delta^+_\infty$, then $\Delta^-_n \sqsubseteq \Delta \sqsubseteq \Delta^+_n$ for all n and thus $\Delta^-_{n+1} \sqsubseteq F(\Delta, \Delta) \sqsubseteq \Delta^+_{n+1}$ for all n. Hence,

$$\Delta^-_\infty \sqsubseteq F(\Delta^+_\infty, \Delta^+_\infty) \sqsubseteq \Delta^+_\infty.$$

Then, as $F(\Delta^+_\infty,\Delta^+_\infty)$ is closed (F preserve closure),

$$\Delta^+_\infty = \overline{\Delta^-_\infty} \sqsubseteq F(\Delta^+_\infty,\Delta^+_\infty) \sqsubseteq \Delta^+_\infty$$

as wanted.

Greatest fixpoint Let Δ be a fixpoint of F. We showed in the preliminary of this proof that $\Delta^-_\infty \sqsubseteq \Delta \sqsubseteq \Delta^+_\infty$. Hence, $\overline{\Delta} = \Delta^+_\infty$. $\square$

3.5 A Reasoning Principle

In order to reason about types, it is not sufficient to characterize the semantic typing Δ_∞ as a fixpoint of F. It is also important to have some kind of induction principle, which takes advantage of the uniqueness of Δ_∞. We are going to state such a principle. We start by remarking that Δ_∞ can also defined by iteration.

LEMMA 3 (ITERATIONS). *Any sequence $(\Delta_n)_{n\in\mathbb{N}}$ of closed semantic candidates such that $\Delta_{n+1} = F(\Delta_n,\Delta_n)$ converges to the semantic typing Δ_∞ in the lattice of closed semantic candidates.*

PROOF. We consider the sequences $(\Delta^-_n)_{n\in\mathbb{N}}$ and $(\Delta^+_n)_{n\in\mathbb{N}}$ defined during the proof of theorem 2. Both sequences converge to Δ_∞ in the lattice of closed semantic candidates. Furthermore, for all $n \in \mathbb{N}$, we have $\Delta^-_n \sqsubseteq \Delta_n \sqsubseteq \Delta^+_n$. So, by the Sandwich theorem, the sequence $(\Delta_n)_{n\in\mathbb{N}}$ converges to Δ_∞. $\square$

The reasoning principle is a direct corollary of this lemma. We say that a semantic candidate Δ is *constant* if for all types σ and σ' we have $\Delta(\sigma) = \Delta(\sigma')$.

THEOREM 4 (REASONING PRINCIPLE). *Let P be a predicate on closed semantic candidates such that:*

- *if Δ is constant, then Δ satisfies P;*
- *if Δ satisfies P, then so does $F(\Delta,\Delta)$;*
- *if a sequence $(\Delta_n)_{n\in\mathbb{N}}$ converges in the lattice of closed semantic candidates, and all Δ_n satisfies P, then their limit also satisfies P.*

Then, the semantic typing Δ_∞ satisfies P.

PROOF. We consider the sequence $(\Delta_n)_{n\in\mathbb{N}}$ defined by $\Delta_0 = \Delta_\mathcal{B}$ and $\Delta_{n+1} = F(\Delta_n,\Delta_n)$. By lemma 3, this sequence converges to Δ_∞. All Δ_n satisfies P by the two first hypotheses. So, their limit Δ^+_∞ also satisfies P, by the third hypothesis. $\square$

4 Types

4.1 Syntax of Types

Types are defined in two steps. First, we define inductively finite patterns called *type patterns*. Then, these patterns are assembled coinductively [8, 19, 22] into possibly infinite trees called *types*. This two-step construction rules out ill-defined types, such as $\tau = \tau \vee \tau$ because $\tau \vee \tau$ is not a pattern. Indeed, any occurrence of a type in a pattern is below a *constructor* $\rightarrow$ or $\times$.

The different type constructions are standard. See section 4.3 for a precise description of their meaning. We assume given a set of type variables α and a single constant κ. Given a set of types τ, we define *type patterns* t inductively by the grammar below.

$$\begin{array}{rcll} t & ::= & \kappa & \text{constant} \\ & | & \tau \times \tau & \text{pair type} \\ & | & \tau \rightarrow \tau & \text{function type} \\ & | & \alpha & \text{type variable} \\ & | & \top & \text{top type} \\ & | & t \wedge t & \text{intersection type} \\ & | & \forall\alpha.t & \text{universal quantification} \\ & | & \bot & \text{bottom type} \\ & | & t \vee t & \text{union type} \\ & | & \exists\alpha.t & \text{existential quantification} \end{array}$$

We write $t(\tau_1,\ldots,\tau_k)$ when the pattern t has leaves τ_1, ..., τ_k, where each τ_i occurs linearly in t. The finite patterns t are assembled coinductively as follows:

$$\tau \quad ::= \quad t(\tau_1,\ldots,\tau_k) \quad \text{coinductively.}$$

By coinduction, every type τ is of the form $t(\tau_1,\ldots,\tau_k)$. So, we can reason inductively on the structure of type patterns, then coinductively on the structure of types. This turns out to be very convenient. Another point is that all the constructions $\rightarrow$, $\times$, $\wedge$, ..., on type patterns lift in the obvious way to constructions on types. This enables to write types like $\tau_1 \rightarrow \tau_2$, $\tau_1 \wedge \tau_2$ or $\forall\alpha.\tau$.

Types are considered modulo renaming of their bound variables. This does not contradict the coinductive definition of types on the alphabet of patterns since, in fact, α-conversion is only a handy presentation of de Bruijn indices. Note also that we don't assume types to be *regular*: types may have an infinite number of distinct subtrees.

Because of free type variables, we cannot directly associate a set of terms to a type τ. Indeed, we need to provide an interpretation for these variables. We use an *environment* ρ, a function from type variables to set of terms such that for all variable α, the set $\rho(\alpha)$ is closed and the inclusions $\mathcal{B} \subseteq \rho(\alpha) \subseteq \mathcal{S}$ hold. Then, a "type" σ of section 3 is actually a pair, written $(\tau)_\rho$, of a type τ and an environment ρ.

4.2 Language Refinements

In order to give a meaning to types, we need to refine our definition of the language: the language has functions, pairs and a constant. More precisely, we must specify the destructors corresponding to these constructions: the application, the pair projections `fst` and `snd`, and a function `isconst` that test whether its argument is the constant κ.

So, we have an *application* function which associates to two terms e and e' another term noted $e\,e'$, and three functions `fst`, `snd` and `isconst` from terms to terms. We don't assume that any of theses function is a term.

Let f be a function from terms to terms. We say that f is *strict* if $f(\mathcal{B}) \subseteq \mathcal{B}$ (or, equivalently, $\mathcal{B} \subseteq f^{-1}(\mathcal{B})$). We say that f is *continuous* if, for all closed set of terms $\overline{\mathcal{E}}$, the set $f^{-1}(\overline{\mathcal{E}})$ is closed.

The functions `fst`, `snd`, and `isconst` are strict and continuous. If $e' \in \mathcal{S}$, then the function which associate to a term e the term $e\,e'$ is strict and continuous. The approximation operators $(\_)^n$ are continuous. (One can prove that they are also strict.)

The function `fst`, `snd` and the application satisfy some "*commutation*" properties with respect to the approximation operators:

- if $(\mathtt{fst}\,e)^n \in \overline{\mathcal{E}}$, then $\mathtt{fst}\,(e)^{n+1} \in \overline{\mathcal{E}}$;
- if $(\mathtt{snd}\,e)^n \in \overline{\mathcal{E}}$, then $\mathtt{snd}\,(e)^{n+1} \in \overline{\mathcal{E}}$;
- if $(e\,(e')^n)^n \in \overline{\mathcal{E}}$, then $(e)^{n+1}\,e' \in \overline{\mathcal{E}}$.

4.3 Definition of the Type Operator

We discuss informally what properties of the semantic typing Δ_∞ we would like to have. Since most properties can be interpreted as (reversible) elimination rules, we take the freedom to write them as such.

- A term e has type κ iff it is safe and it can safely be applied to the function `isconst`:

$$\frac{\Gamma \vdash e : \kappa}{\Gamma \vdash \mathtt{isconst}\,e : \top}$$

- A term e has type $\tau_1 \times \tau_2$ iff it is safe and it yields a term of type τ_1 when applied to `fst` and a term of type τ_2 when applied to `snd`:

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{fst}\,e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{snd}\,e : \tau_2}$$

- A term e has type $\tau_2 \to \tau_1$ iff it is safe and it yields a term of type τ_1 whenever applied to a term of type τ_2:

$$\frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\,e_2 : \tau_1}$$

- A term e has type α in the environment ρ iff it is included in $\rho(\alpha)$.
- A term e has type $\top$ iff it is safe ($\top$ is the greatest type).
- A term e has type $\tau_1 \wedge \tau_2$ iff it both has types τ_1 and τ_2:

$$\frac{\Gamma \vdash e : \tau_1 \wedge \tau_2}{\Gamma \vdash e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \wedge \tau_2}{\Gamma \vdash e : \tau_2}$$

- A term e has type $\forall\alpha.\tau$ in the environment ρ iff it has type τ in any environments derived from ρ by binding the variable α to a closed set of term $\overline{\mathcal{E}}$ containing $\mathcal{B}$ and contained in $\mathcal{S}$:

$$\frac{\Gamma \vdash e : \forall\alpha.\tau}{\Gamma \vdash e : \tau[\tau'/\alpha]}$$

- A term e has type $\bot$ iff it is a bottom term ($\bot$ is the least type).
- A term e has type $\tau_1 \vee \tau_2$ iff it is in the closure of the elements of type either τ_1 or τ_2. Intuitively, the need of a closure comes from the following typing rule:

$$\frac{\Gamma \vdash e_1 : \tau_1 \vee \tau_2 \qquad \Gamma \vdash e_2 : \tau_1 \to \tau \qquad \Gamma \vdash e_2 : \tau_2 \to \tau}{\Gamma \vdash e_2\,e_1 : \tau}$$

  This is a Curry-style counterpart (the terms do not contain any type) to the more usual typing rule [33]:

$$\frac{\Gamma \vdash e_1 : \tau_1 \vee \tau_2 \qquad \Gamma; x : \tau_1 \vdash e_2 : \tau \qquad \Gamma; x : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathtt{case}\;e_1\;\mathtt{of}\;x \Rightarrow e_2 : \tau}$$

  The rule can be read informally as: a term e_1 has type $\tau_1 \vee \tau_2$ if it can be plugged in any "context" that accept any expression of type τ_1 or τ_2. The closure operator is meant to express this kind of contextual property. One may have expected that a term e has type $\tau_1 \vee \tau_2$ if it has either τ_1 or τ_2. But this is clearly not the right definition in the presence of non-determinism.

- A term e has type $\exists\alpha.\tau$ in the environment ρ iff it is in the closure of elements of type τ in any environment derived from ρ by binding the variable α to a closed set of terms $\overline{\mathcal{E}}$ containing $\mathcal{B}$ and contained in $\mathcal{S}$:

$$\frac{\Gamma \vdash e_1 : \exists\alpha.\tau_1 \qquad \Gamma;\alpha \vdash e_2 : \tau_1 \to \tau \qquad \alpha \notin FV(\tau)}{\Gamma \vdash e_2\,e_1 : \tau}$$

  The reason for the closure is similar as for union types.

The operator F associates to a pair (Δ^+,Δ^-) of semantic candidates a semantic candidate Δ defined inductively on type patterns as follows:

$$\begin{array}{lcl}
\Delta\big((\kappa)_\rho\big) & = & \{e \in \mathcal{S} \mid \mathtt{isconst}\,e \in \mathcal{S}\} \\
\Delta\big((\tau_1 \times \tau_2)_\rho\big) & = & \\
\multicolumn{3}{l}{\quad \{e \in \mathcal{S} \mid \mathtt{fst}\,e \in \Delta^+\big((\tau_1)_\rho\big) \wedge \mathtt{snd}\,e \in \Delta^+\big((\tau_2)_\rho\big)\}} \\
\Delta\big((\tau_2 \to \tau_1)_\rho\big) & = & \\
\multicolumn{3}{l}{\quad \{e \in \mathcal{S} \mid \forall e' \in \Delta^-\big((\tau_2)_\rho\big).e\,e' \in \Delta^+\big((\tau_1)_\rho\big)\}} \\
\Delta\big((\alpha)_\rho\big) & = & \rho(\alpha) \\
\Delta\big((\top)_\rho\big) & = & \mathcal{S} \\
\Delta\big((t_1 \wedge t_2)_\rho\big) & = & \Delta\big((t_1)_\rho\big) \cap \Delta\big((t_2)_\rho\big) \\
\Delta\big((\forall\alpha.t)_\rho\big) & = & \bigcap_{\mathcal{B}\subseteq\mathcal{E}\subseteq\mathcal{S}} \Delta\Big((t)_{\rho[\alpha\mapsto\overline{\mathcal{E}}]}\Big) \\
\Delta\big((\bot)_\rho\big) & = & \mathcal{B} \\
\Delta\big((t_1 \vee t_2)_\rho\big) & = & \overline{\Delta\big((t_1)_\rho\big) \cup \Delta\big((t_2)_\rho\big)} \\
\Delta\big((\exists\alpha.t)_\rho\big) & = & \overline{\bigcup_{\mathcal{B}\subseteq\mathcal{E}\subseteq\mathcal{S}} \Delta\Big((t)_{\rho[\alpha\mapsto\overline{\mathcal{E}}]}\Big)}
\end{array}$$

In order to prove the expected properties of F, it can be useful to define it by the following equivalent, but more algebraic, equalities:

$$\begin{array}{lcl}
\Delta\big((\kappa)_\rho\big) & = & \mathcal{S} \cap \mathtt{isconst}^{-1}(\mathcal{S}) \\
\Delta\big((\tau_1 \times \tau_2)_\rho\big) & = & \\
\multicolumn{3}{l}{\qquad \mathcal{S} \cap \mathtt{fst}^{-1}\big(\Delta^+\big((\tau_1)_\rho\big)\big) \cap \mathtt{snd}^{-1}\big(\Delta^+\big((\tau_2)_\rho\big)\big)} \\
\Delta\big((\tau_2 \to \tau_1)_\rho\big) & = & \mathcal{S} \cap \bigcap_{e' \in \Delta^-((\tau_2)_\rho)} \{e \mid e\,e' \in \Delta^+\big((\tau_1)_\rho\big)\}
\end{array}$$

4.4 Properties of the Operator

We now prove that the operator F indeed defines a best fixpoint Δ_∞, by checking all our assumptions of section 3.3.

LEMMA 5 (WELL-DEFINED). *The function $\Delta = F(\Delta^+,\Delta^-)$ is a semantic candidate.*

PROOF SKETCH. It is clear that $\Delta((t)_\rho) \subseteq \mathcal{S}$ for all t and ρ. We show that $\mathcal{B} \subseteq \Delta((t)_\rho)$ by induction on the type pattern t, using the strictness of `isconst`, `fst` and `snd` and the left-strictness of application. $\square$

LEMMA 6 (MONOTONE). *If* $\Delta_1 \sqsubseteq \Delta'_1$ *and* $\Delta'_2 \sqsubseteq \Delta_2$*, then* $F(\Delta_1,\Delta_2) \sqsubseteq F(\Delta'_1,\Delta'_2)$.

PROOF SKETCH. Let $\Delta = F(\Delta_1,\Delta_2)$ and $\Delta' = F(\Delta'_1,\Delta'_2)$. We assume $\Delta_1 \sqsubseteq \Delta'_1$ and $\Delta'_2 \sqsubseteq \Delta_2$, and prove $\Delta((t)_\rho) \subseteq \Delta'((t)_\rho)$ by induction on the type pattern t. All cases are immediate. □

LEMMA 7 (CLOSED). *If* Δ^+ *is closed, then* $F(\Delta^+,\Delta^-)$ *is closed.*

PROOF SKETCH. We show that $F(\Delta^+,\Delta^-)((t)_\rho)$ is closed by induction on the type pattern t. We use the stability of the closure by intersection, the continuity of `fst`, `snd` and `isconst`, and the left-continuity of application. □

REMARK 8 (ADDITIONAL PROPERTIES OF $\lhd^n$). *The following properties hold:*

- $\overline{\mathcal{E}} \lhd^n \overline{\mathcal{E}}$*;*
- *if* $\mathcal{E}_1 \lhd^n \mathcal{E}'_1$ *and* $\mathcal{E}_2 \lhd^n \mathcal{E}'_2$*, then* $(\mathcal{E}_1 \cup \mathcal{E}'_1) \lhd^n (\mathcal{E}'_1 \cup \mathcal{E}'_2)$*;*
- *if* $\mathcal{E}_1 \lhd^n \mathcal{E}'_1$ *and* $\mathcal{E}_2 \lhd^n \mathcal{E}'_2$*, then* $(\mathcal{E}_1 \cap \mathcal{E}'_1) \lhd^n (\mathcal{E}'_1 \cap \mathcal{E}'_2)$*;*
- *if* $\mathcal{E} \lhd^n \mathcal{E}'$*, then* $\overline{\mathcal{E}} \lhd^n \overline{\mathcal{E}'}$*.*

The second and third properties can be generalized to infinite unions and intersections.

PROOF. All but the last property are easy to check. Let us consider the last property. The assumption can be restated as: $\mathcal{E}$ is included in the inverse image of $\mathcal{E}'$ by $(\_)^n$. So, by monotony, $\mathcal{E}$ is also included in the inverse image of $\overline{\mathcal{E}'}$ by $(\_)^n$. As $(\_)^n$ is continuous, this inverse image is closed. So, it contains the closure of $\mathcal{E}$. That is, $\overline{\mathcal{E}} \lhd^n \overline{\mathcal{E}'}$ as wanted. □

LEMMA 9 (CONTRACTIVE). *If* $\Delta \lhd^n \Delta'$ *and* Δ' *is closed, then* $F(\Delta,\Delta') \lhd^{n+1} F(\Delta',\Delta)$.

PROOF SKETCH. We show that $F(\Delta,\Delta')(t) \lhd^{n+1} F(\Delta',\Delta)(t)$ by induction on the type pattern t. We only consider the most interesting case: $t = \tau_2 \to \tau_1$.

Let $e \in F(\Delta,\Delta')\,(\tau_2 \to \tau_1)_\rho$. We want to show that $(e)^{n+1} \in F(\Delta',\Delta)\,(\tau_2 \to \tau_1)_\rho$. First, as $e \in \mathcal{S}$ and $\mathcal{S}$ is closed, we have $(e)^{n+1} \in \mathcal{S}$. Let $e' \in \Delta((\tau_2)_\rho)$. By assumption, $(e')^n \in \Delta'((\tau_2)_\rho)$. Therefore, $e\,(e')^n \in \Delta((\tau_1)_\rho)$. By assumption again, $(e\,(e')^n)^n \in \Delta'((\tau_1)_\rho)$. By commutation, as the semantic candidate Δ' is closed, $(e)^{n+1}\,e' \in \Delta'((\tau_1)_\rho)$ as wanted. □

5 Subtyping Rules

We have now made enough assumption about the language and the types to prove the soundness of the subtyping rules given in figure 1. These rules are standard. The set $\mathrm{FV}(\tau)$ is the set of *free variables* of τ.

These rules may be interpreted inductively or coinductively. Soundness is obvious in the inductive case, since each rule is immediately true when taken in isolation. But we adopt here the coinductive interpretation [8, 22], which captures more subtyping relations.

$$\textsc{Const}\ \ \kappa <: \kappa \qquad \textsc{Pair}\ \ \frac{\tau_1 <: \tau'_1 \qquad \tau_2 <: \tau'_2}{\tau_1 \times \tau_2 <: \tau'_1 \times \tau'_2} \qquad \textsc{Fun}\ \ \frac{\tau'_2 <: \tau_2 \qquad \tau_1 <: \tau'_1}{\tau_2 \to \tau_1 <: \tau'_2 \to \tau'_1}$$

$$\textsc{Var}\ \ \alpha <: \alpha$$

$$\textsc{Top}\ \ \tau <: \top \qquad \textsc{Inter-R}\ \ \frac{\tau <: \tau_1 \qquad \tau <: \tau_2}{\tau <: \tau_1 \wedge \tau_2} \qquad \textsc{Inter-L1}\ \ \frac{\tau_1 <: \tau}{\tau_1 \wedge \tau_2 <: \tau}$$

$$\textsc{Inter-L2}\ \ \frac{\tau_2 <: \tau}{\tau_1 \wedge \tau_2 <: \tau} \qquad \textsc{All-R}\ \ \frac{\tau <: \tau' \qquad \alpha \notin \mathrm{FV}(\tau)}{\tau <: \forall\alpha.\tau'}$$

$$\textsc{Bot}\ \ \bot <: \tau \qquad \textsc{Union-L}\ \ \frac{\tau_1 <: \tau \qquad \tau_2 <: \tau}{\tau_1 \vee \tau_2 <: \tau} \qquad \textsc{Union-R1}\ \ \frac{\tau <: \tau_1}{\tau <: \tau_1 \vee \tau_2}$$

$$\textsc{Union-R2}\ \ \frac{\tau <: \tau_2}{\tau <: \tau_1 \vee \tau_2} \qquad \textsc{Exists-L}\ \ \frac{\tau' <: \tau \qquad \alpha \notin \mathrm{FV}(\tau)}{\exists\alpha.\tau' <: \tau}$$

Figure 1. Subtyping rules

We write $[\![\tau]\!]_\rho$ for $\Delta_\infty((\tau)_\rho)$, where Δ_∞ is the closed fixpoint of the operator F.

THEOREM 10 (SOUNDNESS OF THE SUBTYPING RULES). *If* $\tau <: \tau'$*, then for all environment* ρ *we have* $[\![\tau]\!]_\rho \subseteq [\![\tau']\!]_\rho$.

PROOF SKETCH. This result is proved by applying the Reasoning Principle (theorem 4). In order to use this theorem, we first need to define the predicate P: for all type patterns t and t', for all environment ρ, if $t <: t'$, then $\Delta((t)_\rho) \subseteq \Delta((t')_\rho)$. We then need to check that each hypothesis of theorem 4 holds. Most hypothesis are immediate. We prove:

if Δ is closed and satisfies P, then so does $F(\Delta,\Delta)$

by induction on the type patterns t and t', and by case on the last rule of a derivation of $t <: t'$. □

6 Orthogonality

We show how an appropriate choice of the closure operator implies immediately the continuity properties of section 4.2.

We define the closure operator using a polarity. Suppose we have a set of "*coterms*", together with an *orthogonality* relation $e \perp c$. The relation induces a polarity between terms and coterms, and therefore a closure relation on terms. The first component of the polarity associates to a set of terms $\mathcal{E}$ a set of coterms $\mathcal{E}^\perp = \{c \mid \forall e \in \mathcal{E}. e \perp c\}$; the second component associates to a set of coterms $\mathcal{C}$ a set of terms $\mathcal{C}^\perp = \{e \mid \forall c \in \mathcal{C}. e \perp c\}$.

We now need to choose a set of coterms and an orthogonality relation. The key ingredient for this choice is the following lemma about adjunction. Let f be a function from terms to terms, and g be a function from coterms to coterms. We say that g is an *adjoint* of f iff

$$f(e) \perp c \Leftrightarrow e \perp g(c).$$

$$\textsc{Abs}\quad \lambda x.e \Downarrow \lambda x.e \qquad \textsc{App}\quad \frac{e \Downarrow \lambda x.e_1 \qquad e' \Downarrow v_2 \qquad e_1[v_2/x] \Downarrow \widetilde{v}}{e\,e' \Downarrow \widetilde{v}} \qquad \textsc{Pair}\quad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{(e_1,e_2) \Downarrow (v_1,v_2)} \qquad \textsc{Const}\quad \kappa \Downarrow \kappa$$

$$\textsc{Case-Const}\quad \frac{e_1 \Downarrow \kappa \qquad e_2 \Downarrow \widetilde{v}}{\texttt{case}\; e_1\; e_2\; e_3\; e_4 \Downarrow \widetilde{v}} \qquad \textsc{Case-Pair}\quad \frac{e_1 \Downarrow (v_1,v_2) \qquad e_3 \Downarrow \widetilde{v}}{\texttt{case}\; e_1\; e_2\; e_3\; e_4 \Downarrow \widetilde{v}}$$

$$\textsc{Case-Abs}\quad \frac{e_1 \Downarrow \lambda x.e \qquad e_4 \Downarrow \widetilde{v}}{\texttt{case}\; e_1\; e_2\; e_3\; e_4 \Downarrow \widetilde{v}} \qquad \textsc{Fst}\quad \frac{e \Downarrow (v_1,v_2)}{\texttt{fst}\, e \Downarrow v_1} \qquad \textsc{Snd}\quad \frac{e \Downarrow (v_1,v_2)}{\texttt{snd}\, e \Downarrow v_1} \qquad \textsc{Para-Left}\quad \frac{e_1 \Downarrow \widetilde{v}}{e_1 \| e_2 \Downarrow \widetilde{v}} \qquad \textsc{Para-Right}\quad \frac{e_2 \Downarrow \widetilde{v}}{e_1 \| e_2 \Downarrow \widetilde{v}}$$

$$\textsc{Var-Error}\quad x \Downarrow \texttt{error} \qquad \textsc{App-Error-1}\quad \frac{e \Downarrow \widetilde{v} \qquad \widetilde{v} \neq \lambda x.e}{e\,e' \Downarrow \texttt{error}}$$

$$\textsc{App-Error-2}\quad \frac{e \Downarrow \lambda x.e_1 \qquad e' \Downarrow \texttt{error}}{e\,e' \Downarrow \texttt{error}} \qquad \textsc{Pair-Error-1}\quad \frac{e_1 \Downarrow \texttt{error}}{(e_1,e_2) \Downarrow \texttt{error}} \qquad \textsc{Pair-Error-2}\quad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow \texttt{error}}{(e_1,e_2) \Downarrow \texttt{error}}$$

$$\textsc{Case-Error}\quad \frac{e_1 \Downarrow \texttt{error}}{\texttt{case}\; e_1\; e_2\; e_3\; e_4 \Downarrow \texttt{error}} \qquad \textsc{Fst-Error}\quad \frac{e \Downarrow \widetilde{v} \qquad \widetilde{v} \neq (v_1,v_2)}{\texttt{fst}\, e \Downarrow \texttt{error}}$$

$$\textsc{Snd-Error}\quad \frac{e \Downarrow \widetilde{v} \qquad \widetilde{v} \neq (v_1,v_2)}{\texttt{snd}\, e \Downarrow \texttt{error}}$$

Figure 2. Reduction rules

LEMMA 11. *If a function f has an adjoint g, then it is continuous.*

PROOF. Let f be a function with an adjoint g. Let $\overline{\mathcal{E}}$ be a closed set of terms. The following propositions are equivalent:

$$\begin{array}{ll} e \in f^{-1}(\overline{\mathcal{E}}) & f(e) \in \overline{\mathcal{E}} \\ \forall c \in \mathcal{E}^{\perp}. f(e) \perp c & \forall c \in \mathcal{E}^{\perp}. e \perp g(c) \\ \forall c \in g(\mathcal{E}^{\perp}). e \perp c & e \in (g(\mathcal{E}^{\perp}))^{\perp} \end{array}$$

Therefore, $f^{-1}(\overline{\mathcal{E}}) = (g(\mathcal{E}^{\perp}))^{\perp}$ is closed. □

We therefore choose the coterms and the orthogonality relation so that we have an adjoint for `fst`, `snd`, `isconst`, the application and the approximation operators $(\_)^n$. This yields naturally to the use of contexts:

$$\begin{array}{rcll} \mathrm{C} & ::= & [\_] & \text{hole} \\ & | & \mathrm{C}\,e & \text{application} \\ & | & e\,\mathrm{C} & \text{application} \\ & | & \texttt{fst}\,\mathrm{C} & \text{first projection} \\ & | & \texttt{snd}\,\mathrm{C} & \text{second projection} \\ & | & \texttt{isconst}\,\mathrm{C} & \text{constant tester} \\ & | & (\mathrm{C})^n & \text{approximation} \end{array}$$

Indeed, let us use the following definition of orthogonality:

$$e \perp \mathrm{C} \quad \text{iff} \quad \mathrm{C}[e] \in \mathcal{S}.$$

Then, all functions expected to be continuous clearly have an adjoint (for instance, $\texttt{fst}\,e \perp \mathrm{C}$ iff $e \perp \mathrm{C}[\texttt{fst}\,[\_]]$). Besides, $\mathcal{S}$ is closed: $\mathcal{S} = \{[\_]\}^{\perp}$.

It is interesting to notice that the specification of union types in section 4.3:

$$\Delta\left((t_1 \vee t_2)_\rho\right) = \overline{\Delta\left((t_1)_\rho\right) \cup \Delta\left((t_2)_\rho\right)}$$

can be rewritten equivalently as:

$$\Delta\left((t_1 \vee t_2)_\rho\right)^{\perp} = \Delta\left((t_1)_\rho\right)^{\perp} \cap \Delta\left((t_2)_\rho\right)^{\perp}$$

which is dual to the specification of intersection types:

$$\Delta\left((t_1 \wedge t_2)_\rho\right) = \Delta\left((t_1)_\rho\right) \cap \Delta\left((t_2)_\rho\right)$$

A similar duality also holds for existential and universal quantification.

Interpreting types as closed sets of terms defined by orthogonality ensures that types are maximal, in the sense that, if we take a term e which is not of some type τ, then we can find a context C accepting any term of type τ but rejecting e.

7 A Simple Call-by-value Calculus

In order to verify that the assumptions made in the previous sections make sense, we consider a simple call-by-value calculus and prove that it satisfies these assumptions.

7.1 Definition

The terms of the calculus are given by the grammar below. We assume an infinite set of variable x. There is a single constant κ.

$$\begin{array}{rcll} e & ::= & x & \text{variable} \\ & | & \lambda x.e & \text{abstraction} \\ & | & e\,e & \text{application} \\ & | & \kappa & \text{constant} \\ & | & (e,e) & \text{pair} \\ & | & \texttt{case}\; e\; e\; e\; e & \text{case operator} \\ & | & \texttt{fst}\, e & \text{first projection} \\ & | & \texttt{snd}\, e & \text{second projection} \\ & | & e \| e & \text{non-deterministic choice} \end{array}$$

We give a big step semantics to the language. Indeed, we don't need to capture diverging behaviors: terms that diverge are well-typed. The *values* are abstractions, pairs, the constant and the error:

$$\begin{array}{rcl} v & ::= & \lambda x.e \mid (v,v) \mid \kappa \\ \widetilde{v} & ::= & v \mid \texttt{error} \end{array}$$

Terms and values are considered modulo renaming of bound variables.

The *convergence relation* $e \Downarrow \widetilde{v}$ is defined by the inductive rules of figure 2. We say that a term e *diverge* if there is no value $\widetilde{v}$ such that $e \Downarrow \widetilde{v}$.

7.2 Defining the Semantic Typing

We now show that this calculus satisfies all assumptions of sections 3.1 and 4.2. We first define safe terms and bottom terms. The safe terms are the terms that do not converge to `error`.

$$\mathcal{S} = \{e \mid \neg(e \Downarrow \texttt{error})\}$$

The bottom terms are the terms that diverge.

$$\mathcal{B} = \{e \mid \neg(\exists \widetilde{v}. e \Downarrow \widetilde{v})\}$$

The closure operator is defined by orthogonality using the definition of safe terms and the contexts of section 6. We can then check that the assumption on $\mathcal{S}$ and $\mathcal{B}$ of section 3.1 are satisfied.

LEMMA 12 (PROPERTIES OF $\mathcal{S}$ AND $\mathcal{B}$). *The sets $\mathcal{S}$ and $\mathcal{B}$ are closed, and the set $\mathcal{B}$ is a subset of $\mathcal{S}$.*

PROOF. It is clear that $\mathcal{B} \subseteq \mathcal{S}$ and $\mathcal{S}$ is closed ($\mathcal{S} = ([\_])^{\perp}$). Let us prove that $\mathcal{B}$ is closed. Let e in $\overline{\mathcal{B}}$. We have $\texttt{fst}\,[\_] \in \mathcal{B}^{\perp}$ and $[\_]\,\kappa \in \mathcal{B}^{\perp}$. Therefore, by definition of the closure operator, we have both $\texttt{fst}\,e \in \mathcal{S}$ and $e\,\kappa \in \mathcal{S}$. This is only possible if e diverges. ☐

We now define a number of useful terms:

$$\begin{array}{lcl}
\texttt{diverge} & = & (\lambda x.x\,x)\,(\lambda x.x\,x) \\
\texttt{fail} & = & \kappa\,\kappa \\
\texttt{isconst} & = & \lambda x.\texttt{case}\,x\,\texttt{diverge}\,\texttt{error}\,\texttt{error} \\
\pi_0 & = & \lambda x.\texttt{diverge} \\
\pi_{n+1} & = & \lambda x.\texttt{case}\,x \\
 & & \quad x \\
 & & \quad (\pi_n\,(\texttt{fst}\,x), \pi_n\,(\texttt{snd}\,x)) \\
 & & \quad \lambda y.\pi_n\,(x\,(\pi_n\,y))
\end{array}$$

The term `diverge` is an element of $\mathcal{B}$. The term `fail` is not an element of $\mathcal{S}$ (that is, $\texttt{fail} \Downarrow \texttt{error}$). The term `isconst` diverges when applied to the constant, and fails when applied to any other value. We define the approximation operators $(\_)^n$ by $(e)^n = \pi_n\,e$.

We can then check that the continuity and strictness assumptions of section 4.2 are satisfied.

LEMMA 13 (CONTINUITY AND STRICTNESS). *The operators `fst`, `snd`, `isconst` and $(\_)^n$ are strict and continuous. If $e' \in \mathcal{S}$, then the function which associate to a term e the term $e\,e'$ is also strict and continuous.*

PROOF SKETCH. Continuity is clear by lemma 11. Strictness is also easy. For instance, if $e \in \mathcal{B}$, then we can prove by contradiction that $\texttt{fst}\,e \in \mathcal{B}$. Indeed, there would otherwise exist a value $\widetilde{v}$ such that $\texttt{fst}\,e \Downarrow \widetilde{v}$. We can easily see that this is not the possible by case on a derivation of this proposition. ☐

We now prove the commutation properties of the approximation operators (section 4.2). The proof relies on the following lemma.

LEMMA 14. *Suppose that, for all $\widetilde{v}$, if $e' \Downarrow \widetilde{v}$ then $e \Downarrow \widetilde{v}$. Then, if $e \in \overline{\mathcal{E}}$ then $e' \in \overline{\mathcal{E}}$.*

PROOF SKETCH. Let us assume that $e \in \overline{\mathcal{E}}$. Let $\mathbf{C}$ in $\mathcal{E}^{\perp}$. We want to prove that $e' \perp \mathbf{C}$, that is, $\mathbf{C}[e'] \in \mathcal{S}$, or $\neg(\mathbf{C}[e'] \Downarrow \texttt{error})$. On the other hand, we have $\neg(\mathbf{C}[e] \Downarrow \texttt{error})$. So, it is sufficient to prove that if $\mathbf{C}[e] \Downarrow \texttt{error}$ then $\mathbf{C}[e'] \Downarrow \texttt{error}$.

We prove by induction on contexts the more general property that, for all context $\mathbf{C}$, if $\mathbf{C}[e'] \Downarrow \widetilde{v}$ then $\mathbf{C}[e] \Downarrow \widetilde{v}$. ☐

LEMMA 15 (COMMUTATION). *The following properties of the approximation operator hold:*

- *if $(\texttt{fst}\,e)^n \in \overline{\mathcal{E}}$, then $\texttt{fst}\,(e)^{n+1} \in \overline{\mathcal{E}}$;*
- *if $(\texttt{snd}\,e)^n \in \overline{\mathcal{E}}$, then $\texttt{snd}\,(e)^{n+1} \in \overline{\mathcal{E}}$;*
- *if $(e\,(e')^n)^n \in \overline{\mathcal{E}}$, then $(e)^{n+1}\,e' \in \overline{\mathcal{E}}$.*

PROOF SKETCH. By application of lemma 14. For instance, for the first property, we prove by case on a derivation of $\texttt{fst}\,(e)^{n+1} \Downarrow \widetilde{v}$ that if $\texttt{fst}\,(e)^{n+1} \Downarrow \widetilde{v}$ then $(\texttt{fst}\,e)^n \Downarrow \widetilde{v}$. ☐

We finish by checking the other properties of the approximation operator (section 3.1).

LEMMA 16 (APPROXIMATION). *The following properties of the approximation operators hold:*

- *if $e \in \overline{\mathcal{E}}$, then $(e)^n \in \overline{\mathcal{E}}$*
- *if $(e)^n \in \overline{\mathcal{E}}$ for all $n \in \mathbb{N}$, then $e \in \overline{\mathcal{E}}$.*
- *if $e \in \mathcal{S}$, then $(e)^0 \in \mathcal{B}$;*

The proof of the two first properties require some work. Intuitively, we show that if we insert approximation operators into a term, then we get less errors. Conversely, if a term fails, it still fails when the rank of the added approximation operators is high enough. To be precise, we define an annotation relation $e \rightsquigarrow_n e'$ stating that the term e' is the term e with some inserted approximation operators of rank at least n (figure 3) and prove the two lemmas below.

LEMMA 17. *If $e' \Downarrow \widetilde{v}'$ and $e \rightsquigarrow_0 e'$, then there exists a value $\widetilde{v}$ such that $e \Downarrow \widetilde{v}$ and $\widetilde{v} \rightsquigarrow_0 \widetilde{v}'$.*

$$\frac{e \rightsquigarrow_n e' \qquad n \le n'}{e \rightsquigarrow_n (e')^{n'}} \qquad \frac{\lambda x.e \rightsquigarrow_n \lambda x.e' \qquad n \le n'}{\lambda x.e \rightsquigarrow_n \lambda y.((\lambda x.e')\,(y)^{n'})^{n'}} \qquad x \rightsquigarrow_n x$$

$$\frac{e \rightsquigarrow_n e'}{\lambda x.e \rightsquigarrow_n \lambda x.e'} \qquad \frac{e_1 \rightsquigarrow_n e_1' \qquad e_2 \rightsquigarrow_n e_2'}{e_1\,e_2 \rightsquigarrow_n e_1'\,e_2'} \qquad \kappa \rightsquigarrow_n \kappa$$

$$\frac{\begin{array}{c} e_1 \rightsquigarrow_n e_1' \\ e_2 \rightsquigarrow_n e_2' \end{array}}{(e_1, e_2) \rightsquigarrow_n (e_1', e_2')} \qquad \frac{e \rightsquigarrow_n e'}{\texttt{fst}\,e \rightsquigarrow_n \texttt{fst}\,e'} \qquad \frac{e \rightsquigarrow_n e'}{\texttt{snd}\,e \rightsquigarrow_n \texttt{snd}\,e'}$$

$$\frac{\begin{array}{cc} e_1 \rightsquigarrow_n e_1' & e_2 \rightsquigarrow_n e_2' \\ e_3 \rightsquigarrow_n e_3' & e_4 \rightsquigarrow_n e_4' \end{array}}{\texttt{case}\,e_1\,e_2\,e_3\,e_4 \rightsquigarrow_n \texttt{case}\,e_1'\,e_2'\,e_3'\,e_4'}$$

$$\frac{e_1 \rightsquigarrow_n e_1' \qquad e_2 \rightsquigarrow_n e_2'}{e_1 \| e_2 \rightsquigarrow_n e_1' \| e_2'} \qquad \texttt{error} \rightsquigarrow_n \texttt{error}$$

Figure 3. Annotation of terms and values

$$
\textsc{Var-Access}\ \frac{\Gamma(x)=\tau}{\Gamma\vdash x:\tau}
\qquad
\textsc{App}\ \frac{\Gamma\vdash e_1:\tau_2\rightarrow\tau_1 \quad \Gamma\vdash e_2:\tau_2}{\Gamma\vdash e_1\,e_2:\tau_1}
\qquad
\textsc{Abs}\ \frac{\Gamma;x:\tau_2\vdash e:\tau_1}{\Gamma\vdash\lambda x.e:\tau_2\rightarrow\tau_1}
\qquad
\textsc{Const}\ \Gamma\vdash\kappa:\kappa
\qquad
\textsc{Pair}\ \frac{\Gamma\vdash e_1:\tau_1 \qquad \Gamma\vdash e_2:\tau_2}{\Gamma\vdash(e_1,e_2):\tau_1\times\tau_2}
\qquad
\textsc{Fst}\ \frac{\Gamma\vdash e:\tau_1\times\tau_2}{\Gamma\vdash\mathtt{fst}\,e:\tau_1}
$$

$$
\textsc{Snd}\ \frac{\Gamma\vdash e:\tau_1\times\tau_2}{\Gamma\vdash\mathtt{snd}\,e:\tau_2}
\qquad
\textsc{Para}\ \frac{\Gamma\vdash e_1:\tau \qquad \Gamma\vdash e_2:\tau}{\Gamma\vdash e_1\|e_2:\tau}
\qquad
\textsc{Union-Elim}\ \frac{\Gamma\vdash e_1:\tau_1\vee\tau_2 \quad \Gamma\vdash e_2:\tau_1\rightarrow\tau \quad \Gamma\vdash e_2:\tau_2\rightarrow\tau}{\Gamma\vdash e_2\,e_1:\tau}
\qquad
\textsc{Exists-Elim}\ \frac{\Gamma\vdash e_1:\exists\alpha.\tau_1 \quad \Gamma;\alpha\vdash e_2:\tau_1\rightarrow\tau \quad \alpha\notin FV(\tau)}{\Gamma\vdash e_2\,e_1:\tau}
\qquad
\textsc{Exists-Intro}\ \frac{\Gamma\vdash e:\tau[\tau'/\alpha]}{\Gamma\vdash e:\exists\alpha.\tau}
\qquad
\textsc{Inter-Intro}\ \frac{\Gamma\vdash e:\tau_1 \qquad \Gamma\vdash e:\tau_2}{\Gamma\vdash e:\tau_1\wedge\tau_2}
$$

$$
\textsc{All-Intro}\ \frac{\Gamma;\alpha\vdash e:\tau}{\Gamma\vdash e:\forall\alpha.\tau}
\qquad
\textsc{All-Elim}\ \frac{\Gamma\vdash e:\forall\alpha.\tau'}{\Gamma\vdash e:\tau'[\tau''/\alpha]}
\qquad
\textsc{Sub}\ \frac{\Gamma\vdash e:\tau' \qquad \tau'<:\tau}{\Gamma\vdash e:\tau}
$$

Figure 4. Typing rules

PROOF SKETCH. The proof is by induction on a derivation of $e'\Downarrow\widetilde{v}'$. $\square$

LEMMA 18. *If $e\Downarrow\widetilde{v}$, then there exists $n\in\mathbb{N}$ such that, for all $n'\in\mathbb{N}$, if $e\rightsquigarrow_{n+n'} e'$ then there exists a value $\widetilde{v}'$ such that $e'\Downarrow\widetilde{v}'$ and $\widetilde{v}\rightsquigarrow_{n'}\widetilde{v}'$.*

PROOF SKETCH. The proof is by induction on a derivation of $e\Downarrow\widetilde{v}$. The integer n depends both on the size of the derivation of $e\Downarrow\widetilde{v}$ and on the size of each intermediate value. $\square$

We can now finish the proofs.

PROOF SKETCH OF LEMMA 16.

- Let us assume that $(e)^n\notin\overline{\mathcal{E}}$ and show that $e\notin\overline{\mathcal{E}}$. We know that there exists $C\in\mathcal{E}^\perp$ such that the relation $(e)^n\perp C$ does not hold. Therefore, by definition of $\perp$ and $\mathcal{S}$, $C[(e)^n]\Downarrow\mathtt{error}$. On the other hand, we have $C[e]\rightsquigarrow_0 C[(e)^n]$. So, by lemma 17, there exists a value $\widetilde{v}$ such that $C[e]\Downarrow\widetilde{v}$ and $\widetilde{v}\rightsquigarrow_0\mathtt{error}$. By inspection of the definition of the annotation relation, we see that $\widetilde{v}=\underline{\mathtt{error}}$. Therefore, the relation $e\perp C$ does not hold: $e\notin\overline{\mathcal{E}}$ as wanted.

- The proof of the second property is similar, but relies on lemma 18 instead of lemma 17.

- By inspection of the reduction rules of figure 2, it is clear that if $e\in\mathcal{S}$, then $(e)^0\in\mathcal{B}$. $\square$

8 Typing Rules

We define some typing rules for the calculus of section 7 and prove their soundness. The *typing judgments* J are defined by the following grammar:

$$J \quad ::= \quad \vdash e:\tau \mid x:\tau;J \mid \alpha;J$$

The constructions $x:\tau;J$ and $\alpha;J$ respectively bind x and α in the judgment J. Judgment are considered modulo renaming of bound variables. A *typing environment* is a context in with one can plug a judgment to form another judgment:

$$\Gamma \quad ::= \quad \_ \mid x:\tau;\Gamma \mid \alpha;\Gamma$$

We note $\Gamma(x)$ the type associated to the variable x in the typing environment Γ. The typing rules are given in figure 4.

In order to prove the soundness of the typing rules, we interpret the judgments as logical propositions:

$$
\begin{array}{rcl}
\llbracket\vdash e:\tau\rrbracket_\rho & = & e\in\llbracket\tau\rrbracket_\rho\\
\llbracket x:\tau;J\rrbracket_\rho & = & \forall e'\in\llbracket\tau\rrbracket_\rho.\llbracket J[e'/x]\rrbracket_\rho\\
\llbracket\alpha;J\rrbracket_\rho & = & \forall\mathcal{E}.\mathcal{B}\subseteq\mathcal{E}\subseteq\mathcal{S}\Rightarrow\llbracket J\rrbracket_{\rho[\alpha\mapsto\overline{\mathcal{E}}]}
\end{array}
$$

We write $\llbracket\tau\rrbracket_\rho$ for $\Delta_\infty((\tau)_\rho)$. The substitution $J[e'/x]$ is the usual capture-avoiding substitution.

We make use of the following lemma in order to prove the soundness of the typing rules EXISTS-INTRO and ALL-ELIM.

LEMMA 19 (TYPE SUBSTITUTION). *The following equality hold:*

$$\llbracket\tau[\tau'/\alpha]\rrbracket_\rho=\llbracket\tau\rrbracket_{\rho[\alpha\mapsto\llbracket\tau'\rrbracket_\rho]}$$

PROOF SKETCH. We simultaneously prove by induction on n that for all type τ, we have $\llbracket\tau[\tau'/\alpha]\rrbracket_\rho\lhd^n\llbracket\tau\rrbracket_{\rho[\alpha\mapsto\llbracket\tau'\rrbracket_\rho]}$ and $\llbracket\tau\rrbracket_{\rho[\alpha\mapsto\llbracket\tau'\rrbracket_\rho]}\lhd^n\llbracket\tau[\tau'/\alpha]\rrbracket_\rho$. $\square$

THEOREM 20 (SOUNDNESS OF THE TYPING RULES). *If $\vdash e:\tau$, then $e\in\mathcal{S}$.*

PROOF SKETCH. We prove that if $\Gamma\vdash e:\tau$ then for all environment ρ the proposition $\llbracket\Gamma\vdash e:\tau\rrbracket_\rho$ holds, by considering each typing rule in turn. Then, if $\vdash e:\tau$, we have $e\in\llbracket\tau\rrbracket_\rho\subseteq\mathcal{S}$ for any environment ρ. $\square$

9 Limits of the Approach

9.1 Proof Techniques

It seems natural to replace the rules EXISTS-INTRO and ALL-ELIM by subtyping rules:

$$
\frac{\tau''<:\tau[\tau'/\alpha]}{\tau''<:\exists\alpha.\tau}
\qquad\qquad
\frac{\tau[\tau'/\alpha]<:\tau''}{\forall\alpha.\tau<:\tau''}
$$

These rules would be obviously sound (by lemma 19) if our subtyping relation were defined inductively. Unfortunately, in our coinductive system, we are not able to extend the proof of our soundness theorem 10 to accommodate these two subtyping rules. Indeed, the current proof is by induction on type patterns, but the type pattern $t[\tau'/\alpha]$ is not necessarily "smaller" than the type patterns $\exists\alpha.t$ and $\forall\alpha.t$.

Another point is that the proof of lemma 19 relies on a very precise knowledge of the structure of the semantic typing (it makes uses of the $\lhd^n$ relations). We don't know how to avoid this.

9.2 Approximation Operators

Our results rely deeply on the approximation operators $(\_)^n$. However, these approximation operators do not exist for arbitrary calculi.

For instance, suppose that we replace the two reduction rules App-Error-1 and App-Error-2 by the rules below:

$$\frac{e \Downarrow \texttt{error}}{e\,e' \Downarrow \texttt{error}} \qquad \frac{e \Downarrow v \qquad e' \Downarrow \texttt{error}}{e\,e' \Downarrow \texttt{error}} \qquad \frac{e \Downarrow v \qquad e' \Downarrow v' \qquad v \neq \lambda x.e}{e\,e' \Downarrow \texttt{error}}$$

In other words, we don't require anymore that the first term of an application is a function, as long as the second term diverges. Then, the assumption:

if $(e\,(e')^n)^n \in \overline{\mathcal{E}}$, then $(e)^{n+1}\,e' \in \overline{\mathcal{E}}$.

does not hold for $\overline{\mathcal{E}} = \mathcal{S}$, since the term $(\kappa(\kappa)^0)^0$ diverges, while the term $(\kappa)^1\,\kappa$ converges to $\texttt{error}$.

Similarly, if we extend the calculus with coinductive values, such as a value v_0 such that $v_0 = (v_0, v_0)$, then the assumption:

if $(e)^n \in \overline{\mathcal{E}}$ for all $n \in \mathbb{N}$, then $e \in \overline{\mathcal{E}}$

does not hold for $\overline{\mathcal{E}} = \mathcal{B}$ and $e = v_0$, since $(v_0)^n$ diverges for all n, while v_0 is a value.

In the first case, a workaround is to modify the definition of the operator F for function types to:

$$\begin{array}{l} \Delta\big((\tau_2 \to \tau_1)_\rho\big) \quad = \\ \quad \{e \in \mathcal{S} \,|\, \texttt{isfun}\,e \in \mathcal{S} \wedge \forall e' \in \Delta^-\big((\tau_2)_\rho\big).e\,e' \in \Delta^+\big((\tau_1)_\rho\big)\} \end{array}$$

where $\texttt{isfun} = \lambda x.\texttt{case}\ x\ \texttt{error}\ \texttt{error}\ \texttt{diverge}$. That is, we force terms of type $\tau_2 \to \tau_1$ to reduce to function values. In that way, we only need the following weaker assumption in our proof:

if $\texttt{isfun}\,e \in \mathcal{S}$ and $(e\,(e')^n)^n \in \overline{\mathcal{E}}$, then $(e)^{n+1}\,e' \in \overline{\mathcal{E}}$.

But this workaround is annoying because it rejects the equality $\bot \to \bot = \top$ which is intuitively appealing in this modified calculus. Indeed, the application of any term of type $\top$ to a term of type $\bot$ diverges.

In fact, there seems to be no way out: defining a family of approximation operators necessarily modifies either the semantics of the calculus or of the typing. This approach is taken, for instance, by Dami [13, 14] who adds the approximation operators directly in the language. In the calculus with coinductive values mentioned above, this results in an altered semantics where new values are admitted like a pair $v_1 = (\Omega, \Omega)$, where $\texttt{fst}\,v_1$ diverges (these values are introduced to approximate coinductive values such as v_0). But, then, the type $\bot \times \bot$ is inhabited by non-diverging terms such as v_1, while we would expect to have $\bot \times \bot <: \bot$ in a call-by-value language with coinductive values.

10 Related Work

Models of Recursive Types

MacQueen, Plotkin, and Sethi [31] define the *ideal model for types* by interpreting types as ideals of a domain $\mathbf{V}$, which verifies the isomorphism:

$$\mathbf{V} \cong \mathbf{T} + \mathbf{N} + (\mathbf{V} \to \mathbf{V}) + (\mathbf{V} \times \mathbf{V}) + (\mathbf{V} + \mathbf{V}) + \{\texttt{error}\}_\bot.$$

That is, $\mathbf{V}$ is isomorphic to the coalesced sum of the truth values $\mathbf{T}$, the integers $\mathbf{N}$, the continuous functions from $\mathbf{V}$ to $\mathbf{V}$, the product of $\mathbf{V}$ with itself, the sum of $\mathbf{V}$ with itself, and a value $\texttt{error}$.

An ideal I is a non-empty set of elements of $\mathbf{V}$ which is downward closed and closed under least upper bounds of increasing sequences. The ideals are actually the closed sets of the Scott topology (but the empty set). Our framework can be instantiated by taking $\mathcal{S} = \mathbf{V} \setminus \{\texttt{error}\}$ so that these closed sets coincide with our closed sets of section 6. Indeed, a closed set I of the Scott topology is closed according to our definition: there exists a continuous function f from $\mathbf{V}$ to $\mathbf{V}$ such that

$$\begin{array}{l} f\,e = \bot \text{ if } e \in I \\ f\,e = \texttt{error} \text{ otherwise,} \end{array}$$

and therefore $I = \{f\,[\_]\}^\perp$. Conversely, one can check that our closed sets are downward closed and closed under least upper bounds of increasing sequences.

The three authors specify a metric on the ideals of $\mathbf{V}$, such that the set of ideals is a complete metric space. The denotation of a recursive type is computed as the fixpoint of an operator from ideals to ideals which is shown to be contractive. In our case, this metric can be defined using the $\lhd^n$ relations: $d(I,J) = 0$ if $I = J$; otherwise, $d(I,J) = 2^{-n}$ where n is the greatest integer such that $I \lhd^n J$ and $J \lhd^n I$. The equivalent of our iteration principle (lemma 3) is proved using the Banach fixpoint theorem.

Chroboczek [12] introduces a metric in the same spirit to analyze recursive types in game semantics.

Damm [15] applies the ideal model to study the subtyping of union and intersection types and to prove the soundness and completeness of a subtyping algorithm.

Cartwright [11] defines types semantically as intervals, that is, as pairs of two ideals (I,J) such that $I \subseteq J$. Such a typing provides two pieces of information at the same time: every term $e \in I$ has type (I,J), and every term of type (I,J) is element of J. The interesting point about intervals is that an operator F from ideals to ideals which is monotone in its first argument and anti-monotone in its second argument induces a *monotone* operator from intervals to intervals: this operator associate to an interval (I,J) the interval $(F(I,J), F(J,I))$. Then, by the Knaster-Tarski fixpoint theorem, this monotone operator has a least fixpoint (this correspond to the beginning of our proof of theorem 2). Unfortunately, the two bounds of the fixpoint do not *a priori* coincide. At this point, Cartwright uses the same metric space argument as in the ideal model to establish that the bounds coincide in the case of contractive operators.

In domain theory, a solution $\mathbf{V}$ of a recursive equation $\mathbf{V} = G(\mathbf{V})$ is usually obtained by bi-limit [3] of the sequence defined inductively by $\mathbf{V}_0 = \{\perp\}$ and $\mathbf{V}_{n+1} = G(\mathbf{V}_n)$. We can consider that $\mathbf{V}_n \subseteq \mathbf{V}$ (the domain $\mathbf{V}_n$ is isomorphic to a subset of $\mathbf{V}$) and define a family of *projections* π_n from $\mathbf{V}$ to $\mathbf{V}_{n+1}$. These projections corresponds to our approximation operators. They are central to most works derived from the ideal model. Only Appel and McAllester [5] take a different approach: they give a small-step semantics to their calculus, and limit the number of steps to get an approximation of the behavior of a term.

Pitts [34] is interested in solving general recursive equations on domains. He applies a similar technique as Cartwright, inspired by the work of Freyd [20] on algebraically compact categories, but notices that the coincidence of the two bounds of the limit interval is a consequence of a so-called *minimal invariant property*, corresponding to the fact that the identity is the limit of the projections. Our proof of theorem 2 is based on this very idea.

Dami [14, 13] uses a notion of labelled reduction, inspired by the calculi by Lévy [30], Hyland [28], and Wadsworth [42]. The disadvantage of instrumenting the semantics this way is that it modifies the properties of the initial calculus. Abadi, Pierce and Plotkin[1] realized that the projections can be defined in some well-chosen syntactic calculi, and use this idea to define a faithful semantics of types. This idea of denotable projections appears to be fruitful. It was taken up successfully by Smith, Mason, and Talcott [32, 37], and Birkedal and Harper [6] to transfer approximation techniques from denotational semantics to operational semantics.

In the works of Appel and McAllester [5] and of Dami [14, 13], the semantic typing Δ_∞ does not satisfy the fixpoint property ($\Delta_\infty = F(\Delta_\infty)$), but only a post-fixpoint property ($\Delta_\infty \subseteq F(\Delta_\infty)$) corresponding to soundness. The semantic typing is defined as the greatest lower bound $\Delta_\infty = \inf(\Delta_n)$ of a decreasing sequence of semantic candidates Δ_n defined inductively. The advantage of this approach is that it is easier to define Δ_∞. But this semantic typing Δ_∞ is *a priori* not canonical: it may depend on the way the sequence (Δ_n) is defined. Consequently, if one wants to prove a given property on types, one has to work on the sequence Δ_n, instead of on its limit Δ_∞.

Orthogonality, Continuity

The notion of polarities [18] was introduced by Birkhoff [7]. The notion of closure by orthogonality appeared recently in proof theory, independently in Krivine-style realizability [16] and in Girard's Ludics [23]. Pitts [35] uses a similar notion of closure to express logical relations in operational semantics. But his typed framework remains to be connected to the untyped framework favored by proof theoreticians.

The notion of continuity can be extended from topology to sets X equipped with a "closure" operator c which here is simply a function from $\mathcal{P}(X)$ to $\mathcal{P}(X)$, without any other assumption: $f : X \to Y$ is continuous if $c(f^{-1}(B)) \subseteq f^{-1}(c(B))$ for all $B \subseteq Y$ [38, 39]. In the case of closure operators as they are defined in section 2, this definition is equivalent to our definition of continuity (section 4.2).

11 Future Work

This article is only a first step in a wider program. We thought it was important to start with the "elementary" case of types interpreted as set of terms (that is, unary relations on terms). The next step will be to study PER models and logical relations [2, 4, 9, 10, 35, 36] in the same way.

We assume that the calculus is extensional, and therefore has no side-effect. In particular, some of our typing rules are not sound [17] in presence of references. It will be interesting to see what happen in a calculus with side-effect. A starting point may be different work on reasoning about effects [25, 36, 40, 41].

12 References

[1] M. Abadi, B. Pierce, and G. Plotkin. Faithful ideal models for recursive polymorphic types. *International Journal of Foundations of Computer Science*, 2(1):1–21, Mar. 1991. Summary in Fourth Annual Symposium on Logic in Computer Science, June, 1989.

[2] M. Abadi and G. Plotkin. A per model of polymorphism and recursive types. In J. Mitchell, editor, *5th Annual IEEE Symposium on Logic in Computer Science*, pages 355–365. IEEE Computer Society Press, 1990.

[3] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[4] R. Amadio. Recursion over realizability structures. *Information and Computation*, 91:55–85, 1991.

[5] A. W. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(5):657–683, 2001.

[6] L. Birkedal and R. Harper. Constructing interpretations of recursives types in an operational setting. *Information and Computation*, 155:3–63, 1999.

[7] G. Birkhoff. *Lattice theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, 1940.

[8] M. Brandt and F. Henglein. Coinductive axiomatization of recursive type equality and subtyping. In R. Hindley, editor, *Proc. 3d Int'l Conf. on Typed Lambda Calculi and Applications (TLCA), Nancy, France, April 2-4, 1997*, volume 1210 of *Lecture Notes in Computer Science (LNCS)*, pages 63–81. Springer-Verlag, April 1997.

[9] K. Bruce and J. Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Proc. 19th ACM Symp. on Principles of Programming*, pages 316–327, 1992.

[10] F. Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 164–178, Stresa, Italy, July 1989. Springer-Verlag.

[11] R. Cartwright. Types as intervals. In *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 22–36. ACM Press, 1985.

[12] J. Chroboczek. Subtyping recursive games. In *Proceedings of the Fifth International Conference on Typed Lambda Calculi and Applications (TLCA'01).*, Kraków, Poland, May 2001.

[13] L. Dami. Labelled reductions, runtime errors and operational subsumption. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 782–793. Springer, 1997.

[14] L. Dami. Operational subsumption, an ideal model of subtyping. In A. D. Gordon, A. M. Pitts, and C. Talcott, editors, *HOOTS II Second Workshop on Higher-Order Operational Techniques in Semantics*, volume 10 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2000.

[15] F. Damm. Subtyping with union types, intersection types and recursive types II. Research Report 2259, INRIA Rennes, May 1994.

[16] V. Danos and J.-L. Krivine. Disjunctive tautologies and synchronisation schemes. In *Computer Science Logic'00*, volume 1862 of *Lecture Notes in Computer Science*, pages 292–301. Springer, 2000.

[17] R. Davies and F. Pfenning. Intersection types and computational effects. In ICFP '00 [29], pages 198–208.

[18] M. Erné, J. Koslowski, A. Melton, and G. E. Strecker. A primer on Galois connections. In A. R. Todd, editor, *Papers on general topology and applications (Madison, WI, 1991)*, volume 704 of *Annals of the New York Academy of Sciences*, pages 103–125, New York, 1993. New York Acad. Sci.

[19] M. P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Information and Computation*, 127(2):186–198, 1996.

[20] P. J. Freyd. Algebraically complete categories. In A. Carboni, M. C. Pedicchio, and G. Rosolini, editors, *Proceedings of the 1990 Como Category Theory Conference*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer–Verlag, 1991.

[21] A. Frisch, G. Castagna, and V. Benzaken. Semantic subtyping. In *17th IEEE Symposium on Logic in Computer Science*, pages 137–146. IEEE Computer Society Press, 2002.

[22] V. Gapeyev, M. Levin, and B. Pierce. Recursive subtyping revealed. In ICFP '00 [29]. To appear in *Journal of Functional Programming*.

[23] J.-Y. Girard. Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science*, 11(3):301–506, June 2001.

[24] A. D. Gordon and A. M. Pitts, editors. *Higher Order Operational Techniques in Semantics*. Publications of the Newton Institute. Cambridge University Press, 1998.

[25] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proceedings of the 11th Annual Conference of the European Association for Computer Science Logic (CSL'02)*, volume 2471 of *Lecture Notes in Computer Science*, pages 553–568. Springer-Verlag, Sept. 2002.

[26] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Transactions on Internet Technology*, 2002. Submitted for publication.

[27] H. Hosoya, J. Vouillon, and B. C. Pierce. Regular expression types for XML. *ACM Transactions on Programming Languages and Systems (TOPLAS)*. To appear; short version in ICFP 2000.

[28] J. M. E. Hyland. A syntactic characterization of the equality in some models of the λ-calculus. *Journal of the London Mathematical Society*, 12(2):361–370, 1976.

[29] *Proceedings of the the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, Montréal, Canada, Sept. 2000. ACM Press.

[30] J.-J. Lévy. An algebraic interpretation of the lambda beta K-calculus; and an application of a labelled lambda-calculus. *Theoretical Computer Science*, 2(1):97–114, June 1976.

[31] D. MacQueen, G. Plotkin, and R. Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71(1-2):95–130, 1986.

[32] I. A. Mason, S. F. Smith, and C. L. Talcott. From operational semantics to domain theory. *Information and Computation*, 128(1):26–47, 1996.

[33] B. C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, Feb. 1991.

[34] A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

[35] A. M. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in computer Science*, 10:321–359, 2000.

[36] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In Gordon and Pitts [24], pages 227–273.

[37] S. F. Smith. The coverage of operational semantics. In Gordon and Pitts [24], pages 307–346.

[38] B. M. R. Stadler and P. F. Stadler. Basic properties of closure spaces, 2002. Supplemental material for [39].

[39] B. M. R. Stadler and P. F. Stadler. Generalized topological spaces in evolutionary theory and combinatorial chemistry. *J. Chem. Inf. Comput. Sci.*, 42:577–585, 2002. Proceedings MCC 2001, Dubrovnik.

[40] I. Stark. Names, equations, relations: practical ways to reason about new. *Fundamenta Informaticae*, 33(4):369–396, 1998.

[41] C. Talcott. Reasoning about functions with effects. In Gordon and Pitts [24], pages 347–390.

[42] C. P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. *SIAM Journal on Computing*, 5(3):488–521, Sept. 1976.