Semantically Linear Programming Languages

Luca Paolini
Dipartimento di Informatica
Università di Torino
10149 Torino, Italy
paolini@di.unito.it

Mauro Piccolo*
Dipartimento di Informatica
Università di Torino
10149 Torino, Italy
piccolo@di.unito.it

ABSTRACT
We propose a paradigmatic programming language (called \$\ell$PCF) which is linear in a semantic sense. \$\ell$PCF is not syntactically linear, namely its programs can contain more than one occurrences of the same variable. We give an interpretation of \$\ell$PCF into a model of linear coherence spaces and we show that such semantics is fully abstract with respect to our language. Furthermore, we discuss the independence of new syntactical operators and we address the universality problem.

Keywords
Linear Functions, Coherence Spaces, PCF, Continuous, Stable and Strongly Stable Functions.

1. INTRODUCTION
Coherence Spaces [19] are the result of the deep analysis of stable semantics [7] done by Jean-Yves Girard. These spaces provide a pleasant decomposition of stable functions via linear functions and exponential domain constructors. Such a decomposition is patently reflected in linear logic syntax. Since its inception, linear logic has inspired the introduction of many formal languages with sometimes different goals: resource-conscious evaluation, categorical language, implicit computational complexity, and so on.

The starting point is the least full subcategory of coherence spaces, including the infinite flat domain (representing natural numbers) and the coherence spaces of linear functions between domains in the model itself. We avoided the use of exponential domain constructors, since we want focus on linearity. We sought a programming language able to program the functions (with respect to a standard interpretation) of a the considered category (playing the role of model).

We propose \$\ell$PCF, acronym of Semantically-linear-PCF, a language based on a syntactic restriction of PCF togheter with a new operator. The considered linear model is indeed fully abstract for our language. It should be clear that the considered linear model is not correct (w.r.t. standard interpretations) for almost all languages inspired to linear logic, since they are able to programs all stable functions between coherence spaces. More precisely, the considered model is correct for the “linear core” of these languages. A preliminary version of \$\ell$PCF, without full abstraction and independence properties, was presented in [17].

Linear functions are strict in all their arguments. Except under conditions, high-order variables (having an arrow as type) occur just once in terms. The conditional requires the use of the same high-order variables in both its branches. So strictness for higher-order abstractions (terms abstracting an arrow-typed variable) can be assured by the operational evaluation. The case of first-order abstractions (terms abstracting ground variables) is more complex. Let $\lambda x. n$ be the non-strict function associating the natural number $n$ to all possible inputs (also undefined ones), namely an erasing function. Linear endo-functions on a flat domain are all the stable functions, except the erasing ones. In other words, the union of the set $\{\lambda x. n | n \in \mathbb{N}\}$ of erasing functions with the set of linear function $\mathbb{N} \to \mathbb{N}$ is one-to-one with the space $!\mathbb{N} \Rightarrow \mathbb{N}$ of stable functions. In order to represent all functions in $\mathbb{N} \to \mathbb{N}$, in our language, we do not put any constraints on the occurrences of ground variables. They can occur more than once or zero times. The strictness on first-order abstractions (abstracting ground variables) is provided by applying a call-by-value parameter passing policy. Note that this liberality on the management of ground variables morally entails that we may also use a high-order variable more than once, provided that we always apply it
to the same sequence of arguments. More precisely, the result of the evaluation of a ground term (possibly containing high-order subterms) can be either erased or duplicated. We introduced a third kind of variables, namely stable variables. They are used only in order to program a linear recursion. These key points behind $\ell$PCF imply the correctness of our purely linear model.

This analysis teaches us that linearity can be considered in many respects. Three main kinds of linearity emerge: syntactical, operational and denotational. The syntactical one claims a linear use of variables in terms. This kind of linearity is sometimes considered the computational counterpart of the linear logic linearity. $\ell$PCF is not linear in this syntactical sense. However, $\ell$PCF is linear in a denotational sense, since only purely linear functions can be defined in the language, without any kind of exponentiation. Finally, if redexes are not duplicated during the evaluation, then a notion of operational linearity arises. $\ell$PCF is not endowed with a linear operational evaluation, although the restriction $\ell$PCF to terms without fixpoints (its finitary fragment) is operationally linear. Operational linearity is related to the notion of simple term [24] in $\lambda$-calculus, suggesting a more general linearity than the operational one, unrelated from a specific strategy.

The main results of this paper are the very simple proof of full abstraction and the proof of independence of new construct of $\ell$PCF. A very general motivation behind our results is that linearity which is embedded in our language is more generous than the strict syntactical one, so it can be used in order to improve resource-conscious results related to Logical $\ell$PCF. $\ell$PCF is Turing-Complete. However, type-respecting recursive functions in StPCF are strictly less than that of the languages StPCF [29] and PCF+$\eta$ [26]. In other words, linear functions between two coherence spaces are less than stable functions between the same domains. Since the syntactical constraints of $\ell$PCF forbid useless duplications of redexes, we are utterly convinced that $\ell$PCF can be fruitfully exploited in research fields like optimal evaluation (see for instance [2, 31, 34]), implicit computational complexity (see for instance [4, 5, 14]), linear computation (see for instance [1, 13]). Further motivations are theoretical. Our study is relevant from an higher-type computability point of view, since we introduce new operators. In [30], we are studying processes behind our programming language by translating programs in calculus of Solos [25]. Such a translation is related to the description of game-semantics done in [6, 23, 35], by using the $\tau$-calculus [33]. We plan also to study correspondences between Ludics [20] and the linear model, considered above.

**Outline of the paper.**

Section 2 presents $\ell$PCF, its operational semantics. Moreover, we discuss the calculus behind the language. Section 3 tackles some remarks on which $\ell$, a syntactical operator of $\ell$PCF. Section 4 recalls basic notions on coherence spaces, together with the subsections proving interpretation, correctness and completeness. Section 5 discuss further extensions of $\ell$PCF which are correct for our linear model, conclusions and conjectures. In its subsections, we introduce strongly-stable models and a Boolean version of a second-order Gustave-like OR operator.

**2. A SEMANTICALLY LINEAR PROGRAMMING LANGUAGE**

$\ell$PCF is a PCF-like language with implicit truth-values which are coded on integers (zero codes “true” while any other numeral stands for “false”).

As customary $\rightarrow\sigma$ associates to right. Hence $\sigma_1 \rightarrow\sigma_2 \rightarrow\sigma_3$ is an abbreviation for $(\sigma_1 \rightarrow\sigma_2) \rightarrow\sigma_3$). Furthermore, it is easy to see that all types $\tau$ have shape $\tau_1 \rightarrow\cdots \tau_n \rightarrow\iota$, for some type $\tau_1, \ldots, \tau_n$ where $n \ge 0$.

Let $\text{Var}^\sigma$, $\text{SVar}^\sigma$ be numerable sets of variables of type $\sigma$. The set of high-order variables is $\text{HVar} = \bigcup_{\sigma, \tau} \text{Var}^{\sigma \rightarrow \tau}$ and the whole set of variables is $\text{Var} = \text{Var}^\iota \cup \text{HVar} \cup \text{SVar}$. Letters $x^\sigma, y^\sigma, z^\sigma, \ldots$ range over variables in $\text{Var}^\sigma$ while $f^\iota, f^\tau, f^\sigma, \ldots$ ranges over stable variables, namely variables in $\text{SVar}^\sigma$. Last, $\kappa$ will denote any kind of variables. Latin letters $\overline{m}, \overline{N}, \ldots$ range over terms.

**Definition 1.** The set $\mathbb{T}$ of linear types is defined as follows: $\sigma, \tau ::= \iota \mid (\sigma \rightarrow \tau)$ where $\iota$ is the only ground type and $\sigma, \tau, \ldots$ are metavariables ranging over types.

Note that both ground and stable variables can occur freely in our language. They belong to distinct kinds only for sake of simplicity. Except for the $\ell$-if construction, high-order variables are treated linearly.

Sometimes types will be omitted when they are clear from the context or uninteresting (note that given types of all variables of a term $M$, there is a unique $\sigma$ such that $M^\sigma$). Sometimes, parentheses are omitted, always by respecting the following conventions: application associates to the left and application binds more tightly than abstraction, i.e. $\lambda x.\overline{m} = (\lambda x.\overline{m}) \overline{n}$.

**Definition 2.** Let $\Gamma \subseteq \text{HVar}$. Typed terms of $\ell$PCF are defined by using a type assignment proving judgments of the shape $\Gamma \vdash M : \sigma$, in Table 1.

In order to define the evaluation of the language, we need pairing and projections operators on natural numbers. If $m, n \in \mathbb{N}$ then $m^{\overline{n}}(2n+1)-1$ is our pairing function. Projections from $z \in \mathbb{N}$ can be defined by functions

$$\min((3y) \le z (z = < x, y >)) \quad \min((z < x, y >)).$$

Such functions induce a bijection, details can be found either, in p.41,p.73 of [12] or in p.47 of [15].
Definition 3. The evaluation relation $\triangledown \subseteq \mathcal{P} \times \mathcal{N}$ is the effective relation inductively defined by the rules of Table 2. If there exists a numeral $n$ such that $M \downarrow n$, then we say that $M$ converges, and we write $M \uparrow$, otherwise we say that it diverges, and we write $M \uparrow$.

A restriction of $\Sigma$PCF endowed with an operational semantics, has been presented in [17]. The relation $\uparrow$ implements a call-by-value parameter passing policy in the ground case and call-by-name parameter passing policy in the high-order case.

Syntax deserves some discussion. We remark that $\text{pred}$ is a partial operator, namely $\text{pred } \mathbf{0}$ diverges. The evaluation of $\ell \text{iff } L' \cdot L' \cdot K'$ asks to evaluate exactly one subterm between $L'$ and $R'$. The evaluation of which? $M$ is a pattern for infinite rules, similarly to rules of the operator $\text{of Plotkin}$ [32]. Note that which? $M \downarrow$ if and only if $M(\lambda x . x) \downarrow$; moreover, linearity assures that $M$ applies its argument $\lambda x . x$ exactly to a unique numeral $k$ in Table 2. A deterministic evaluation can be formalized likewise that of the operator strict? presented in [29]. Since first-order strict stable functions are linear, we need to add a fixpoint operator to our language. Unfortunately, the least fixpoint of a linear function is always the bottom of the considered domain, because strictness. In order to overcome this problem we recall the fixpoint theorem [22].

**Theorem 1.** Let $D$ be a cpo. If $f : D \to D$ is continuous then it has a least fixpoint $\text{fix}(f) \in D$.

We introduced a special kind of variables, named stable variables. Those variables are used in order to program continuous (w.r.t. stable order) non-strict functions from a linear coherence space $L$ to itself, so their fixpoints will be elements of $L$. Syntactically, a stable variable will be used without linear constraints. We do not permit to $\lambda$-abstract those variables, since they will be used only in order to obtain fixpoints. Since Turing-Completeness was proved for a restriction of $\Sigma$PCF (without which?) in [17], we can program pairings and projections in the remaining language as done in Table 2.

**Lemma 1.** Let $M', N' \in \Sigma$PCF.

1. If $\text{HV}(N') \cap \text{FV}(N') = \emptyset$ and $x' \in \text{HV}(N')$ then $M'[\cdot/ x'] \in \Sigma$PCF.

2. If $FV(N') \subseteq \text{SVar} \cup \text{Var}$ then $M'[N'/ F'] \in \Sigma$PCF.

3. If $N$ does not contain high-order free variables and $x_1, \ldots, x_n \in \text{Var}$ then $M[N/x_1, \ldots, N/x_n] \in \Sigma$PCF.

**Proof.** Easy, by induction on terms. □

Lemma 1 implies that evaluation is well-defined.

**Definition 4.** Let $[\sigma]$ be a special constant of type $\sigma$. The set of $\sigma$-context $\text{Ctx}_\sigma$ is generated by the following grammar:

$$
\begin{align*}
C[\sigma] & ::= [\sigma] | x' \ | \ell \text{iff } C[\cdot] C[\cdot] \mid (\lambda x . C[\cdot]) \mid (C[\cdot] C[\cdot]) \mid \mu F . C[\cdot] \\
C[\mathbb{N}] & \text{denotes the result obtained by replacing all the occurrences of } [\sigma] \text{ in the context } C[\cdot] \text{ by the term } \mathbb{N} \text{ and by allowing the capture of its free variables.}
\end{align*}
$$

Clearly, $\mathbb{N} \in \Sigma$PCF and $C[\sigma] \in \text{Ctx}_\sigma$ doesn’t imply that $C[\mathbb{N}] \in \Sigma$PCF.

**Definition 5.** (OPERATIONAL EQUIVALENCE). Let $M', N' \in \Sigma$PCF.

1. $M \leq_{\sigma} N$ whenever, for all $C[\sigma]$ s.t. $C[M], C[N] \in \mathcal{P}$, if $C[M] \downarrow n$ then $C[N] \downarrow n$.

2. $M \approx_{\sigma} N$ if and only if $M \leq_{\sigma} N$ and $N \leq_{\sigma} M$.

It is easy to verify that $\leq_{\sigma}$ is a preorder while $\approx_{\sigma}$ is a congruence. It is useful to name some terms. We put $\mathbb{O} = \mu F . F'$. Moreover, if $\sigma_0 = \mu_1 \cdots \mu_n \rightarrow \sigma$ for some $n \in \mathbb{N}$, then $\mathbb{O}^{\sigma_0} = \lambda x_1 \cdots x_n . x_1 \cdots x_n . \ell \text{iff}(\ell\text{iff} . \cdots \text{iff} . \ell\text{iff} . \cdots \text{iff} . \ell\text{iff} . \cdots \text{iff} \ell\text{iff} . \cdots \text{iff} . \ell\text{iff} . \cdots \text{iff} . \ell\text{iff}).$

By using $\mathbb{O}^{\sigma}$ it is possible to define approximants of a term having shape $\mu F . M'$ as follows,

$$
\begin{align*}
\mu F . M' & = \mathbb{O}^{\sigma}, \\
\mu^{n+1} F . M' & = M[\mu^{n} F . M' / F].
\end{align*}
$$

**Lemma 2.** Let $\mathbb{M}_0, \ldots, \mathbb{M}_n$ be a sequence of closed terms $(m \geq 0)$.
1. $\Omega^{\varepsilon_{n-1}} \cdots \varepsilon_1 \varepsilon_0$ is a program
   and $\Omega^{\varepsilon_{n-1}} \cdots \varepsilon_1 \varepsilon_0 \uparrow_s$.

2. Let $(\mu \cdot P^n) \Omega_0 \cdots \Omega_n$ be a program. $(\mu \cdot P^n) \Omega_0 \cdots \Omega_n \uparrow n$
   if and only if $(\mu \cdot P^{n+1} \cdot P^n) \Omega_0 \cdots \Omega_n \uparrow n$, for some $k \in \mathbb{N}$.

2.1 A Semantically Linear Lambda-Calculus

It is good to make a clear distinction between reduction systems and programming languages, as remarked in [28, pp.283]. A reduction system is a (formal) language together with some rewriting rules. A programming language may be regarded as a reduction system with some additional features. It is an essential feature of a programming language to be equipped with a specified reduction strategy.

In this Section, we present the calculus behind S\(\Pi\)CF. Let \(HVF(M) = \text{FV}(M) \cap \text{HVar}\). The language of S\(\Pi\)CF and their sets of free variables (we use \(\text{FV}\) as notation) are mutually (re-)defined in Table 3.

In Definition 4, we defined the notion of context. In what follows, we need some restricted notions of context. We call \(\mu\)-
lazy context, contexts without holes under \(\mu\)-abstractions. Moreover, we define \(\ell\text{-if-lazy context}\), ranging over \(\mathcal{W}[\cdot]\), as follows:

$$\mathcal{W}[\cdot] := | \cdot | \alpha^\varepsilon | f^\varepsilon | \Theta | \mathbf{0} | \text{succ} | \text{pred} \text{ which?}$$

$$| \text{\text{if } } \mathcal{W}[\cdot] | \text{L } | (\lambda x. \mathcal{W}[\cdot]) | (\mathcal{W}[\cdot]\mathcal{W}[\cdot]) | (\mu \cdot \mathcal{W}[\cdot]) .$$

Definition 6. (Reduction Rules) We denote $\rightarrow_{\mathcal{W}}$ the firing everywhere in any context, of one of the following rules:

$$
\begin{align*}
(\lambda x^\varepsilon \cdots M)|_{x^\varepsilon} & \rightarrow_{\varepsilon \beta} M|_{x^\varepsilon} \\
(\lambda x. M)|_{x^\varepsilon} & \rightarrow_{\varepsilon \beta} M|_{x^\varepsilon} \\
\mu \cdot M & \rightarrow_{\beta} \mu \cdot M \quad \text{if } \text{pred succ } \\
\mu \text{ if } L \text{ R } & \rightarrow_{\beta} L \\
\text{which? } (\lambda x^\varepsilon \cdots | \mathcal{W}[\cdot]|_{x^\varepsilon} ) & \rightarrow_{\varepsilon \beta} | \mathcal{W}[\cdot]|_{x^\varepsilon} \\
\text{where } | \mathcal{W}[\cdot]|_{x^\varepsilon} & \text{ is a } \ell\text{-if-lazy context.}
\end{align*}
$$

We denote $\leadsto$ the binary relation on terms obtained as union of the relation-rules $\rightarrow_{\varepsilon \beta}, \rightarrow_{\alpha \beta}, \alpha \rightarrow_{\varepsilon \beta}$ (without any context-closure). Moreover, we denote $\rightarrow_{\theta \varepsilon}$ and $\rightarrow_{\theta \beta}$ respectively, the reflexive and transitive closure of $\rightarrow_{\epsilon}$ and the reflexive, symmetric and transitive closure of $\rightarrow_{\beta}$. Redexes of our calculus are the left-sides of relation-rules defined above.

Clearly, only well-typed \(\ell\text{-if}\)-lazy contexts play a role in the definition of which?-rule. Always in the which?-rule, the first-order variable $f$ abstracted is the same variable filling the hole of the context $\mathcal{W}[\cdot]$; therefore, we mean that $f$ cannot be bound from the context. Whence, we can restrict $\mathcal{W}[\cdot]$ to be \(\mu\)-lazy, since the \(\mu\)-constraints of S\(\Pi\)CF prevents a $\mu$-
abstraction on the hole. Last but not least, we remark that the linearity on $f$ assure that the hole is unique.

As done in [8], it is easy to prove properties as the post-
position of $\delta$-rules in a sequence of reductions, the confluence and a standardization theorem.

**Lemma 3.** Let $\mathcal{W}, \mathcal{N}$ be terms of S\(\Pi\)CF such that $\mathcal{M} \leadsto \mathcal{W}$ and $C[\mathcal{M}, \mathcal{N}] \in \mathcal{P}$, for a context $C[\cdot]$. If $C[\mathcal{M}] \downarrow \mathcal{N}$ then $C[\mathcal{M}] \downarrow \mathcal{N}$.\]

**Proof.** The proof can be done by induction on the shape of contexts, by considering in each case the last rule applied on the derivation proving $C[\mathcal{N}] \downarrow \mathcal{N}$.\]

Let $\mathcal{M} \in \mathcal{P}$ and $\mathcal{N}$ be a numeral, if $\mathcal{M} \rightarrow_{\theta \varepsilon} \mathcal{N}$ according to the leftmost strategy then $\mathcal{M} \downarrow \mathcal{N}$. Something stronger holds, since the confluence of our calculus.

**Theorem 2.** $\mathcal{M} \downarrow \mathcal{N}$ if and only if $\mathcal{M} \rightarrow_{\theta \varepsilon} \mathcal{N}$, for all term $\mathcal{M}$.\]

**Proof.** $\mathcal{M} \downarrow \mathcal{N}$ implies $\mathcal{M} \rightarrow_{\theta \varepsilon} \mathcal{N}$ is trivial. The other direction follows by induction on the number of reduction steps of $\mathcal{M} \rightarrow_{\theta \varepsilon} \mathcal{N}$. The case of zero steps is immediate. The inductive case follows by Lemma 3.\]

Following the suggestion presented in the Introduction, we remark that S\(\Pi\)CF is not syntactically linear but it is denotationally linear, moreover its finitary fragment (i.e. terms avoiding use of the recursion operator) is operationally linear. A term $\mathcal{M}$ is simple (see [24]), if in no reduction of $\mathcal{M}$, a redex is multiplied. Although S\(\Pi\)CF is operationally linear in the finitary fragment, there are S\(\Pi\)CF-terms non simple. For example, $\mathcal{M} = (\lambda x. \text{if } x \mathcal{N} \mathcal{M} \mathcal{N} \mathcal{M} \mathcal{N} \mathcal{N} \mathcal{M})$ and $\mathcal{N} = (\lambda x. \lambda y. x)\mathcal{N}$ are both S\(\Pi\)CF-terms, but $\mathcal{M}$ is not simple. In fact $\mathcal{M} \rightarrow_{\theta \varepsilon} \text{if } x \mathcal{N} \mathcal{M} \mathcal{N} \mathcal{M} \mathcal{N} \mathcal{N} \mathcal{M}$. We note that simple terms of $\lambda$-calculus

Table 2: Operational Semantics of S\(\Pi\)CF.
suggest a fourth kind of linearity, namely linear reduction (i.e. all terms of the considered calculus are simple).

In [30], we introduce einSolos, i.e. a typed process calculus based on the calculus of solos [25]. We use einSolos in order to express computational processes generated by the calculus presented here. We define an interpretation of StPCF on einSolos and we discuss how to process redexes of StPCF in a parallel way. More precisely, we show how to simulate any (single) reduction of StPCF in einSolos (no reduction strategy is forced in the interpretation). Afterward, we prove that a suitable observational equivalence between processes is correct w.r.t the operational semantics of StPCF [29]. Likewise, we can use which to simulate exceptions in a linear setting.

3. THE **which**? OPERATOR

A novelty of StPCF is which?. This higher order operator deserves some discussions.

3.1 Programming with **which**?

The operator which? of StPCF plays a role similar to strict? of StPCF [29]. Likewise, we can use which? to define some useful primitives in StPCF in order to deal with a simple kind of exceptions in a linear setting.

First of all, let us introduce the following operator called @which?: \((i \rightarrow i) \rightarrow i \mapsto (i \rightarrow i) \rightarrow i\) with the following operational semantics.

\[
\frac{\lambda x'.\mathit{if}(\mathit{pred} \ldots \mathit{pred} \ x) (\mathit{R}) \ O') \ \downarrow \ n \ \ [n, k] \ \downarrow \ \Sigma \ \ @\text{which?} \ M(i \rightarrow i) \rightarrow i \ \downarrow \ \Sigma
\]

Note that @which? \(M \downarrow \) if and only if \(M \downarrow \). This control operator gives back the result of the application of the functional \(M\) with the function \(N\) together with the unique numeral passed by \(M\) as argument to \(N\).

If \(M(i \rightarrow i) \rightarrow i\) is a term then @which? \(M(\lambda x'.x)\) behaves in the same way as which? \(M\). So @which? may appear more powerful than which?. However the term

\[
\lambda f(i \rightarrow i).\mathit{which?} (\lambda h^{-1}.f(\lambda x'.g(h(x)))
\]

behaves in the same way as @which?. Therefore, by replacing which? with @which? in StPCF, we program the same linear functions.

3.2 Adding **which**? to PCF-like languages

If we add which? together with the operational rule of Table 2 to PCF, then which? may return a non-deterministic result when applied to non-linear functions. For instance, if \(R = \lambda f^{-1}.\mathit{if}(f)\ O') then both which?R \(\uparrow [3,5]\) and which?R \(\uparrow [5,3]\). However, we can add the operator duwh? to PCF together the same operational rule of which? in Table 2 with the following additional hypothesis,

\[
\forall x \leq k \ M(\lambda x'.\mathit{if}(\mathit{pred} \ldots \mathit{pred} \ x) \ \downarrow \ O') \ \uparrow
\]

This hypothesis asks simply, for the minimum \(k\) satisfying the rule of which?, so the determinism is recovered. Such an hypothesis is inane for which? in StPCF. Unfortunately, the given evaluation rule for duwh? is not effective. However, an effective evaluation of duwh? can be formalized likewise that of the operator strict? [29].

It is easy to verify that duwh? can be programmed in PCF + strict? and so in StPCF [29]. Recall that strict?\((i \rightarrow i) \rightarrow i\) checks whether the function \(M(i \rightarrow i) \rightarrow i\) is strict if \(M(i \rightarrow i) \rightarrow i\) is strict then the evaluation of strict?M returns \(\emptyset\), while if \(M(i \rightarrow i) \rightarrow i\) is not strict but \(M(i \rightarrow i) \rightarrow i\) converges then strict?M returns \(1\). Recall that duwh?M \(\downarrow\) if and only if \(M(\lambda x'.x) \ \downarrow\). Let \(=\) be a program simulating the ground equality (a formal definition is at the begin of Subsection 4.4), it is easy to check that

\[
\begin{align*}
T &= \lambda f(i \rightarrow i) \rightarrow i'.\mathit{strict?} (\lambda y'.f(\lambda x'.if (x = z) z (if y \ z z)))
\end{align*}
\]

tests whether \(f\) passes \(x'\) as argument of \(\lambda x'.z\). Thus duwh? can be translated in the following term of PCF + strict?.

\[
\lambda f(i \rightarrow i).\mathit{which?} (\lambda h^{-1}.f(\mathit{if}(x \ \downarrow (y \ z z)))
\]

Note that duwh? is stable and also strongly stable, since PCF + strict? is (see [29]).

We can also compare duwh? with the catch operator, which is present in the language SPCF [11]. catch \(x'\) in \(M\) binds the occurrences of \(x\) in \(M\). Informally, it asks the evaluation of \(M\), if the computation of \(M\) asks the evaluation of the variable \(x'\) then the computation of catch \(x'\) in \(M\) terminates giving \(\emptyset\). Otherwise, if the computation of \(M\) terminates on a numeral \(n\) without using \(x\), then catch \(x'\) in \(M\) returns \(\mathit{succ}(n)\). We can write a term of SPCF having the same behavior of duwh? using catch, in the following way. Clearly the term

\[
T' = \mathit{catch} \ y \in f(\lambda x'.if (x = z) z y)
\]

evaluates to \(\emptyset\) (i.e.

<table>
<thead>
<tr>
<th>Term</th>
<th>Condition</th>
<th>Free Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\emptyset)</td>
<td>(\mathit{HFV}(\emptyset) = \emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\mathit{succ}^{-1})</td>
<td>(\mathit{HFV}(\mathit{succ}) = \emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(\mathit{pred}^{-1})</td>
<td>(\mathit{HFV}(\mathit{pred}) = \emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>which? ((i \rightarrow i) \rightarrow i)</td>
<td>(\mathit{HFV}(\mathit{which?}) = \emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

Table 3: Syntax of Semantically-linear PCF.
causes an error) if \( f \) does not pass \( x \) as argument of \( \lambda z'.z \). Otherwise it evaluates to the successor of the result of the application of \( f \) to \( \lambda z'.z \). So the term
\[
\lambda t^{(\iota \to \iota)} \cdot (\mu t^{(\iota \to \iota)} \cdot \lambda x'. (\lambda w'. \text{if } w \not\preceq (x. \text{pred}(w)) \text{ then } T' \text{ else } 0)) \ni 0
\]
behaves like \( \text{dwh} \).

We can conclude that \( \text{which} \) can be used in a fruitful way, in order to manage some kind of “linear” exceptions in \( \$\text{PCF} \). Note that \( \text{which} \) is able to perform observations that are subtly linear: in fact \( \text{which} \) gives back at once the parameter passed by \( M \) during its evaluation with the identity function together with the result of the evaluation, in a unique evaluation of the such an application. Thus, the evaluation is done by respecting operational linearity.

### 3.3 Independence of \( \text{which} \)?

We will show that \( \text{which} \) can be interpreted in a linear function, so as a corollary, we will obtain that \( \text{which} \) is also a Scott-continuous and stable function (see [3, pp. 29]). Now, we ask ourselves if there is a program of \( \text{PCF} \) having the same semantics of our \( \text{dwh} \). The answer is negative! We sketch the proof.

If \( M \) and \( N \) are programs then \( M \ni N \) is an abbreviation for the application of the following term to \( M \) and \( N \):
\[
\mu t^{(\iota \to \iota)} \cdot \lambda x'. \text{if } x \not\preceq 0 \text{ then } (\text{if } y \ni (f \preceq \text{pred}(x))(\text{pred}(y))).
\]
Namely, \( \ni \) simulates the equality between numerals. Let
\[
F^* = \lambda t^{(\iota \to \iota)} \cdot \lambda w'. \text{if } w \ni 0 \text{ then } 0 \text{ else } 0,
\]
\[
G_1 = \lambda w^{(\iota \to \iota)} \cdot \lambda x'. \text{if } ((w F^*_2) \ni 0) \text{ then } 0 \text{ else } 0,
\]
\[
G_2 = \lambda w^{(\iota \to \iota)} \cdot \text{if } ((w F^*_2) \ni 0) \text{ then } 0 \text{ else } 0,
\]
be \( \text{PCF} \) terms. Clearly the evaluation of \( G_1 \cdot \text{dwh} \) should diverge, while \( G_2 \cdot \text{dwh} \) should return \( 0 \).

First, we prove that \( M_1, M_2 \) are operationally equivalent in \( \text{PCF} \) extended with \( \text{por} \) (see [22, pp.181]). We use notation and Scott-Continuous model of \( \text{PCF} \) presented in [22]. We recall only that \( \sqsubseteq \) denotes the extensional order. Let \( F^*_2 = \lambda t^{(\iota \to \iota)} \cdot \lambda w'. \text{if } w \ni 0 \text{ then } 0 \text{ else } 0 \) is easy to check that
\[
C[F^*_2] \sqsubseteq C[F^*], \quad C[F^*] \sqcup C[F^*_2] = C[F^*].
\]
For all \( d \) element of the Scott-domain \( C[N_1 \to N_1] \to N_1 \), if \( d \in C[F^*_2] \) and \( d \circ C[F^*] = 0 \) then \( 0, \ni \sqsubseteq d \circ C[F^*] \) by monotonicity. Thus we can state that, such an element \( d \) cannot exist. Easily, it follows that \( M_1, M_2 \) are equivalent in the standard Scott-Continuous model. The correctness of this model implies that \( M_1, M_2 \) are operationally equivalent in \( \text{PCF} + \text{por} \).

**Proposition 1.** There exists no program in \( \text{PCF} + \text{por} \) (and so in \( \text{PCF} \)) having the same semantics of \( \text{dwh} \).

Note that programs of \( \$\text{PCF} \) that do not use \( \text{which} \) are program of \( \text{PCF} \) (by writing \( \text{if} \) in place of \( \text{lit} \)). Thus the above proposition implies the following corollary:

**Corollary 1.** \( \text{which} \) is not syntactic sugar for \( \$\text{PCF} \).

### 4. Linear Model

We are interested in the least full subcategory of coherence spaces, including \( N \) and the coherence spaces of linear functions between domains in the category itself. After the introduction coherence spaces, we prove that the considered model is fully abstract with respect to \( \$\text{PCF} \).

#### 4.1 Coherence Spaces

Coherence spaces are a simple framework for Berry’s stable functions [7], developed by Girard [19]. Proof details can be found in [21].

**Definition 7.** A coherence space \( X \) is a pair \( \langle |X|, \sqsubseteq X \rangle \) where \( |X| \) is a set of tokens called the web of \( X \) and \( \sqsubseteq X \) is a reflexive and symmetric relation between tokens of \( |X| \) called the coherence relation \( \sqsubseteq \) on \( X \). A clique \( x \) of \( X \) is a subset of \( |X| \) containing pairwise coherent tokens. The set of cliques of \( X \) is denoted \( Cl(X) \).

If \( X \) is a coherence space then \( Cl(X) \) form a cpo with respect to the set-theoretical inclusion. In particular,
- \( \emptyset \in Cl(X) \) and \( \{a\} \in Cl(X) \), for each \( a \in |X| \),
- if \( y \subseteq x \) and \( y \in Cl(X) \) then \( y \in Cl(X) \),
- if \( D \subseteq Cl(X) \) is directed then \( \cup D \in Cl(X) \).

**Definition 8.** Let \( X \) and \( Y \) be coherence spaces and \( f : Cl(X) \rightarrow Cl(Y) \) be a monotone function.

- \( f \) is continuous whenever \( \forall x \in Cl(X) \forall b \in f(x) \exists x_0 \subseteq f(x) \text{ s.t. } b \in f(x_0) \).
- \( f \) is stable whenever \( \forall x \in Cl(X) \forall b \in f(x) \exists x_0 \subseteq f(x) \text{ such that } b \in f(x_0) \) and \( \forall x' \subseteq x \), \( b \in f(x') \) then \( x_0 \subseteq x' \).
- \( f \) is linear whenever \( \forall x \in Cl(X) \forall b \in f(x) \exists a \in x \text{ s.t. } b \in f(\{a\}) \).

Some well-known characterizations hold for these functions.

**Lemma 4.** Let \( X \) and \( Y \) be coherence spaces.

- If \( f : Cl(X) \rightarrow Cl(Y) \) is a monotone function then \( f \) is continuous if and only if \( f(\cup D) = \cup \{f(x) : x \in D\} \), for each directed \( D \subseteq Cl(X) \).
- If \( f : Cl(X) \rightarrow Cl(Y) \) is a continuous function then \( f \) is stable if and only if \( \forall x, x' \in Cl(X), x \sqcup x' \in Cl(X) \) implies \( f(x \sqcup x') = f(x) \sqcap f(x') \).

\(^1\)The strict incoherence \( \preceq X \) is the complementary relation of \( \sqsubseteq X \); the incoherence \( \succeq x \) is the union of relations \( \preceq X \) and \( \preceq X \); the strict coherence \( \sim X \) is the complementary relation of \( \succeq x \).
If $f : \mathcal{CL}(X) \rightarrow \mathcal{CL}(Y)$ is a stable function then $f$ is linear if and only if $f(x) = \bigcup_{a \in x} f([a])$ and $f(\emptyset) = \emptyset$.

Linear functions can be represented as cliques.

Definition 9. Let $X$ and $Y$ be coherence spaces. $X \rightarrow Y$ is the coherence space having $|X \rightarrow Y| = |X| \times |Y|$ as web, while $(a, b) \subseteq X \rightarrow Y$ $(a', b')$ is defined as,

$$a =_X a' \text{ implies } b =_Y b' \text{ and } a =_X a' \text{ implies } b =_Y b'$$

Let $X$ and $Y$ be coherence spaces. The trace of a linear function $f : \mathcal{CL}(X) \rightarrow \mathcal{CL}(Y)$ is the set defined as follows:

$$\mathcal{TR}(f) = \{(a, b) \in |X| \times |Y| \mid b \in f([a])\}$$

Let $X$ and $Y$ be coherence spaces, $t \in \mathcal{CL}(X \rightarrow Y)$ and $x \in \mathcal{CL}(X)$. Let us define the map $\mathcal{F}(t) : \mathcal{CL}(X) \rightarrow \mathcal{CL}(Y)$ to be the function such that

$$\mathcal{F}(t)(x) = \{b \in |Y| \mid \exists a \in x, (a, b) \in t\}$$

Lemma 5. If $f : \mathcal{CL}(X) \rightarrow \mathcal{CL}(Y)$ is a linear function then $\mathcal{TR}(f) \subseteq \mathcal{CL}(X \rightarrow Y)$. If $t \in \mathcal{CL}(X \rightarrow Y)$ then $\mathcal{F}(t) : \mathcal{CL}(X) \rightarrow \mathcal{CL}(Y)$ is a linear function.

The basis of our model is the infinite flat domain. Let $\mathbb{N}$ denote the space of natural numbers, namely $\{|\mathbb{N}|, \subseteq \mathbb{N}\}$ such that $|\mathbb{N}| = \mathbb{N}$ and $\{a \rightarrow \tau\}$ be a coherence space; for sake of simplicity, its tokens will be wrote as $(a_1, ..., a_m, b)$ where $a_1 \in |\mathbb{N}|$ for each $i \leq m$ and $b \in |\mathbb{N}|$.

An environment $\rho$ is a function that associates to each variable $\sigma'$ a clique in $\mathcal{CL}(|\sigma|)$. The set of environments is denoted by $\text{Env}$. If $d \in \mathcal{CL}(|\sigma|)$ and $\sigma' \in \text{Var}$ then $\rho(\sigma := d) : \mathbb{N}$ is the environment such that, $\rho(\sigma := d)(\sigma') = d$, but if $\sigma' \notin \sigma$ then $\rho(\sigma := d)(\sigma') = \rho(\sigma')$.

Definition 10. Let $\mathbb{M}^*, \mathbb{N}^*$ be SPCF and $\rho \in \text{Env}$. The interpretation $[\cdot]$ is defined in Table 4.

$\mathbb{M}^* \rightarrow \mathbb{N}^*$ (denotationally equivalent) if and only if $[\mathbb{M}^*] = [\mathbb{N}^*]$ for each $\rho$. The interpretation of closed terms is invariant with respect to environments, thus in such cases the environment can be omitted. The basic properties of a lambda-model are recalled in Lemma 6.

Note that $\mathcal{F}$ is defined in Equation 2 and $\text{fix}$ is defined in Theorem 1.

Lemma 6. Let $\mathbb{M}^*, \mathbb{N}^* \in \text{SPCF}$ and $\rho, \rho' \in \text{Env}$.

1. If $\rho(\sigma) \subseteq \rho'(\sigma)$ for each $\sigma \in \text{FV}(\mathbb{M})$, then $[\mathbb{M}] \rho \subseteq [\mathbb{M}] \rho'$.

2. If $\mathbb{M}^*[\mathbb{N}/\sigma'] \in \text{SPCF}$ then $[\mathbb{M}^*[\mathbb{N}/\sigma']] = [\mathbb{M}] \rho \rho'$.

3. If $\sigma = \tau, C[\sigma] \in \text{Ctxt}_\text{a}$, $[\mathbb{M}] \rho, [\mathbb{N}] \rho \in \mathbb{P}$ and $[\mathbb{M}] \rho = [\mathbb{N}] \rho$ then $[C[\mathbb{M}] \rho] = [C[\mathbb{N}] \rho]$.

Proof. 1. and 2. are easy, by induction on the structure of $\mathbb{M}^*$. 3. follows by induction on the structure of $\mathbb{C}[\sigma]$.

Lemma 7. $[\mu_{\sigma} \mathbb{M}^*] \rho = \bigcup_{n \in \mathbb{N}} [\mu^{n+1} \mathbb{M}^*] \rho$, for all $\rho \in \mathbb{P}$.

Proof. Since $[\mu^{n+1} \mathbb{M}^*] \rho = [\mathbb{M}^*] \rho$, the proof is easy.

Lemma 8. $\mathbb{M} \in \mathbb{P}$ if and only if $\mathbb{M} \downarrow [\mathbb{N}]$.

Proof. The proof can be done by induction on the derivation proving $\mathbb{M} \downarrow [\mathbb{N}]$, likewise to similar proofs in [29, 32].

4.3 Adequacy and Correctness

The denotational semantics is said to be adequate when $[\mathbb{M}] = [\mathbb{M}]$ and $\mathbb{M} \equiv \mathbb{N}$ are logically equivalent for any program $\mathbb{M}$, numeral $\mathbb{N}$. We straightforward adapt a proof of Plotkin [32] for Scott-continuous domains, based on a computability argument in Tait style.

Definition 11. The “computability predicate” is defined by the following cases.

- Case $\text{FV}(\mathbb{M}) = \emptyset$.
  - Subcase $\sigma = \mathbb{I}$. $\text{Comp}(\mathbb{M})$ if and only if $[\mathbb{M}] \rho = [\mathbb{N}] \rho$ implies $\mathbb{M} \equiv [\mathbb{N}]$.
  - Subcase $\sigma = \mu \rightarrow \tau$. $\text{Comp}(\mathbb{M} \rightarrow \sigma)$ if and only if $\text{Comp}(\mathbb{M} \rightarrow \tau)$ for each closed $\mathbb{M}^*$ such that $\text{Comp}(\mathbb{M}^*)$.

- Case $\text{FV}(\mathbb{M}) = \{s_1, ..., s_n\}$, for some $n \geq 1$.
  $\text{Comp}(\mathbb{M})$ if and only if $\text{Comp}(\mathbb{M}^*)$ for each closed $\mathbb{M}^*$ such that $\text{Comp}(\mathbb{M}^*)$.

Lemma 9 states an equivalent formulation of computability predicate.

Lemma 9. $\mathbb{M}^* \rightarrow \sigma^* \rightarrow \mathbb{N}^* \in \text{SPCF}$ and $\text{FV}(\mathbb{M}) = \{s_1, ..., s_n\}$, for each closed $\mathbb{M}^*$ and $\mathbb{P}^*$ such that $\text{Comp}(\mathbb{M}^*)$ and $\text{Comp}(\mathbb{P}^*)$.

Lemma 10. If $\mathbb{M}^* \in \text{SPCF}$ then $\text{Comp}(\mathbb{M}^*)$. 

\[\boxed{\text{Note: } \mathcal{F} \text{ is defined in Equation 2 and } \text{fix} \text{ is defined in Theorem 1.}}\]
Table 4: Interpretation Map.

\[
\begin{align*}
\text{[Q]} \rho &= \{0\} & \text{[succ}^{n}\text{]} \rho &= \{(n, n + 1) \mid n \in \mathbb{N}\} & \text{[pred}^{n}\text{]} \rho &= \{(n, n - 1) \mid n \in \mathbb{N}\} \\
\text{where?}^{(i < j)} \rho &= \{((n, n), (n, r)) \mid n, r \in \mathbb{N}\} \\
\text{[(if } M^{i} \text{ N}^{j} \text{ L}^{k} \text{)}] \rho &= \{n \in \mathbb{N} \mid M^{i} \rho = \{0\} \land N^{j} \rho = \{n\}\} \cup \{n \in \mathbb{N} \mid M^{i} \rho = \{m + 1\} \land L^{k} \rho = \{n\}, m \in \mathbb{N}\} \\
\text{[x]} \rho &= \rho(x) & \text{[\lambda x. M]} \rho &= \{(a_0, b) \in [\sigma] \times [\tau] \mid b \in M^{\sigma} \rho \mid x := \{a_0\}\} \\
\text{[M}^{\omega} \text{N}^{\omega} \text{]} \rho &= F(M^{\omega} \rho \mid N^{\omega} \rho) & \text{[\mu x. M}^{\omega} \text{]} \rho &= \text{fix}(\lambda x. [M]\rho[f := x])
\end{align*}
\]

As usual the adequacy implies the correctness. Note that correctness implies that our terms are strict in all arguments, for all orders.

**Theorem 3.** The linear interpretation is correct for \( \text{S}\text{PCF} \).

**Proof.** Let \( M^{i} \) and \( N^{j} \) such that \( [M]_\rho = [N]_\rho \), for each environment \( \rho \in \text{Env} \). Let \( C[\sigma] \) such that \( [C]_\rho \in \mathbb{P} \). If \( [C] \Downarrow \mathbb{n} \) for some value \( \mathbb{n} \), then \( [C[N]] = [n] \) by Lemma 8. Since \( [C[N]] = [C[M]] = [\mathbb{n}] \) by Lemma 6, \( C[N] \Downarrow \mathbb{n} \) by adequacy. By definition of operational equivalence the proof is done.

\section*{4.4 Completeness and Full Abstraction}

If \( \sigma \in \mathbb{T} \) then, as usual, its order \( \text{order}(\sigma) \) is defined by \( \text{order}(i) = 0 \) and \( \text{order}(\sigma_1 \sim \ldots \sim \sigma_k \sim \iota) = 1 + \max\{\text{order}(\sigma_i) \mid i \in [1, k]\} \).

We show that, all tokens of cohesive spaces are definable. If \( M \) and \( N \) are closed ground terms then \( M = N \) is an abbreviation for the application of

\[
\mu \text{L } \iota \lambda x. (\text{if } y \text{ then } (\text{if } f \text{ then } x) \text{ else } y) \text{ to } M \text{ and } N.
\]

It is easy to check that

\[
[M] \equiv [N] = \begin{cases} 
0 & \text{if } M = m = [N], \\
1 & \text{if } M = m \neq n = [N], \\
\emptyset & \text{otherwise}.
\end{cases}
\]

Let \( N_0 \) and \( N_1 \) be an abbreviation for the term

\[
\text{if } N_0 (\text{if } N_1 0 1) (\text{if } N_1 1 1).
\]

**Lemma 11** (Token Definability). Let \( \sigma \in \mathbb{T} \). If \( a \in [\sigma] \) then there is a closed term \( M^{\omega} \) such that \( [M^{\omega}] = \{a\} \).

**Proof.** The proof is done by induction on the order of \( \sigma \).

The case \( \text{order}(\sigma) = 0 \) is trivial. Assume \( \text{order}(\sigma) = 1 \), thus \( \sigma = \iota_1 \sim \ldots \sim \iota_k \sim \iota \). Hence, given \( a \in [\sigma] \), \( a \) has the shape \((n_1, \ldots, n_k, n)\) where \( n_1, \ldots, n_k, n \in \mathbb{N} \). If \( M^{\omega} = \lambda x_1 \ldots x_k \),

\[
\text{if } ((x_1 \equiv \mathbb{n}_1) \text{ and } (x_2 \equiv \mathbb{n}_2) \text{ and } \ldots \text{ and } (x_k \equiv \mathbb{n}_k)) \in \Omega^{\iota}
\]

then \( [M^{\omega}] = \{a\} \), for all \( \rho \).

Finally, assume \( \sigma = \iota_1 \sim \ldots \sim \iota_k \sim \iota \) where \( \text{order}(\sigma_i) > 1 \) and \( \sigma_i = \iota_1 \sim \ldots \sim \iota_{k_i} \sim \iota \) for \( 1 \leq i \leq k \). If \( a \in [\sigma] \)
then a has shape $((a_1, \ldots, a_k, n_1), \ldots, (a_1', \ldots, a_k', n_k), n')$ where $n_1, \ldots, n_k, n' \in \mathbb{N}$ and $a_i \in [\tau_i]$, for each $i \in [1, k]$ and $j \in [1, h_i]$. By inductive hypothesis, for all $i \in [1, k]$ and $j \in [1, h_i]$, there exists a closed term $M_{ij}^\ast$ such that $\llbracket M_{ij}^\ast \rrbracket = \{a_i^\ast\}$. Thus,

$$\lambda x_1^{\sigma_1} \cdots \lambda x_k^{\sigma_k}. \phi f \left( \frac{(x_1 M_{ij}^\ast \cdots M_{jk}^\ast \ldots) \equiv n_i}{\mathfrak{N}_i} \right) \left( \text{ and } \cdots \right) \frac{(x_k M_{ij}^\ast \cdots M_{jk}^\ast \ldots) \equiv n_k}{\mathfrak{N}_k}$$

is the the closed term defining the considered token.

**Lemma 12.** (Separability). Let $\sigma \in \mathcal{T}$. For all distinct $f, g \in C([\sigma])$ there exists a closed term $M^{\sigma_{\ast} f} \in \wp PCF$ such that $\mathcal{F}(M^{\sigma_{\ast} f}) (f) \neq \mathcal{F}(M^{\sigma_{\ast} f}) (g)$.

**Proof.** Let $\sigma = \sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow \iota$. Since $f \neq g$ there exists $a$ such that $a \in f$ but $a \not\in g$. Assume $a = (a_1, \ldots, a_n, n)$ (where $a_i \in [\sigma_i], \ldots, a_n \in [\sigma_n], n \in \mathbb{N}$). By lemma 11, for each $i \in [1, k]$ there is $M_i \in \wp PCF$ such that $\llbracket M_i \rrbracket = a_i$. By choosing $M_i^{\ast} \ast \phi f (x \equiv n) \in \mathfrak{N}_i$, the proof follows. $\square$

Note that in [29, 32] the separability needs the definability of finite elements, while in our setting the token definability is sufficient. The Separability Lemma implies completeness and full abstraction, as in [29, 32]. Clearly which $\ast$ is not necessary in order to obtain the full abstraction, but it is necessary in order to define all finite elements of the considered model.

**Theorem 4.** (Completeness).
If $N_1 \approx_{\sigma} N_2$ then $\llbracket N_1 \rrbracket = \llbracket N_2 \rrbracket$.

**Proof.** Let us prove the contraposition: let us assume $\llbracket N_1 \rrbracket \neq \llbracket N_2 \rrbracket$, for any environment $\rho$. By definable separability, there exists $M^{\ast} \ast \phi f$ such that $\mathcal{F}(M^{\ast} \ast \phi f) \llbracket N_1 \rrbracket = n_1 \neq \mathcal{F}(M^{\ast} \ast \phi f) \llbracket N_2 \rrbracket = n_2$. By adequacy, $M^{\ast} \ast \phi f \Downarrow N_1$ and $M^{\ast} \ast \phi f \Downarrow N_2$. So $N_1 \neq_{\sigma} N_2$. $\square$

**Corollary 2.** (Full abstraction).
$N_1 \approx_{\sigma} N_2$ if and only if $\llbracket N_1 \rrbracket = \llbracket N_2 \rrbracket$.

5. **LINEAR EXTENSIONS OF $\wp PCF$**

Denotational semantics is usually proposed as a tool to study the equivalence of programs. However, a further use of denotational models is as an abstract tool, assisting us in the comparison and analysis of whole programming languages endowed with different type-respecting computational power (such approach, was first suggested in [26] and pursued in [29]). Such approach, already used in Subsection 3.3, is crucial also in what follows.

**Definition.** A model is universal for a language when every effective element (of domains the interpretation of types) is definable by a closed term of the language [27].

Linear functions are also Scott-continuous with respect to extensional order, see [3, pp. 29]. In order to explore the possible linear extensions of $\wp PCF$ we start by considering some well-known extensions of $PCF$.

It has been proved in [32] that Scott-continuous domains are fully abstract and universal for $PCF^{++}$, namely $PCF$ extended with a parallel conditional pif and an existential operator $\exists$.

It has been shown that stable domains (di-domains, qualitative domains, coherence spaces) give a fully abstract model [29] for $StPCF$, a syntactical extension of $PCF$. The language $StPCF$ is obtained by extending $PCF$ with two operators: gor and strict?. gor corresponds to a Gustave-like or function, while strict? corresponds to a non extensional-monotone function.

However strict? does not respect the Scott-continuity, while pif and $\exists$ do not respect the stability. Whence, they cannot belong to our language. On the other hand, gor respect both Scott-continuity and stability, but it’s not strict, and therefore it’s not linear.

In [26] $PCF$ has been extended with the operator $H$. $PCF + H$ is universal for the strongly stable model [9]. Note that strict? is strongly stable [29], thus it can be defined in $PCF + H$. Thus $H$ does not respect the extensionality and, then, the linearity of our model. Moreover, since $d\exists h^\ast$ can be defined in $PCF + strict?$ our language can program only strongly stable functions.

Unfortunately, $\wp PCF$ does not enjoy of the universality w.r.t. our model. Actually, it is not sufficient in order to obtain the definability of finite cliques.

$$\left\{ \begin{array}{ll}
(0,0) & (0,1) \\
(1,0) & (1,1) \\
(1,1) & (1,1)
\end{array} \right.$$
5.1 Strongly stable model

Definition 13. A qualitative domain \([Q, Q]\) is a pair \(|Q|, Q\) where \(|Q|\) is a set (called the web) and \(Q\) is a subset of \(\wp(|Q|)\) satisfying the following conditions:

- \(\emptyset \in Q\) and, if \(a \in |Q|\) then \(\{a\} \in Q\)
- if \(x \in Q\) and if \(y \subseteq x\) then \(y \in Q\)
- if \(D \subseteq Q\) is directed with respect to inclusion, the \(\bigcup D \in Q\)

The elements of \(Q\) are called states of the qualitative domain, and the qualitative domain itself will also be denoted \(Q\).

Property 1. A qualitative domain \(Q\) is a coherence space when for all \(u \subseteq |Q|\), if for all \(a, b \in u, \{a, b\} \in Q\) then \(u \in Q\).

If \((D, \leq)\) is a poset and \(A, B \subseteq D\) then we say that \(A\) is Egli-Milner smaller than \(B\) (written \(A \subseteq EM B\)) if

\[\forall x \in A, \exists y \in B, x \leq y \quad \text{and} \quad \forall y \in B, \exists x \in A, x \leq y\]

Definition 14. (QUALITATIVE DOMAIN WITH COHESION). A qualitative domain with cohesion is a pair \((Q, C(Q))\) where \(Q\) is a qualitative domain while \(C(Q)\) is a subset of \(\wp_{fin}(Q)\) such that

- if \(x \in Q\) then \(\{x\} \in C(Q)\)
- if \(A \in C(Q)\) and \(B \subseteq EM A\) then \(B \in C(Q)\)
- if \(D_1, \ldots, D_n\) is a family of directed subset of \(Q\) such that for any \(d_1 \in D_1, \ldots, d_n \in D_n\) the set \(\{d_1, \ldots, d_n\} \in C(Q)\) then \(\bigvee D_1, \ldots, D_n \in C(Q)\)

Intuitively, the notion of coherence corresponds to the notion of linear coherence in a sequential structure, see [10, 16].

Table 5: A Second Order Gustave-OR.

<table>
<thead>
<tr>
<th>(P_1 \uparrow 0)</th>
<th>(P_1 \uparrow 0)</th>
<th>(P_2 \uparrow 1)</th>
<th>(P_2 \uparrow 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\cong (\mathcal{G}_0))</td>
<td>(\cong (\mathcal{G}_1))</td>
<td>(\cong (\mathcal{G}_2))</td>
<td>(\cong (\mathcal{G}_3))</td>
</tr>
</tbody>
</table>

\(\cong (\mathcal{G}_1)\) \(\cong (\mathcal{G}_2)\) \(\cong (\mathcal{G}_3)\)

\(\cong (\mathcal{G}_0)\)

Table 6: A boolean version of a Second Order Gustave-OR.

\[b\mathcal{G}_k_{1, k_2, k_3, k_4}(f_1, f_2, f_3) = \begin{cases} k_1 & \text{if } f_1(f_2) = \text{tt}, f_2(f_3) = \text{ff} \\ k_2 & \text{if } f_1(f_2) = \text{ff}, f_2(f_3) = \text{tt} \\ k_3 & \text{if } f_1(f_2) = \text{tt}, f_2(f_3) = \text{ff} \\ k_4 & \text{if } f_1(f_2) = \text{ff}, f_2(f_3) = \text{ff} \\ \bot & \text{otherwise} \end{cases}\]

where \(k_1, k_2, k_3, k_4 \in \{\text{tt}, \text{ff}\}\)

Example 1. Let us consider the qualitative domain with coherence \((B, C(B))\) of booleans,

- \([B] = \{\text{tt}, \text{ff}\}\)
- \(B = \{\emptyset, \{\text{tt}\}, \{\text{ff}\}\}\)
- \(C(B) = \{\emptyset, \{\text{tt}\}, \{\text{ff}\}, \{\emptyset, \{\text{ff}\}\}\}

Morphisms between qualitative domains adapts the definition for coherence space. Let \(X\) and \(Y\) be qualitative domains,

- If \(f : X \rightarrow Y\) is a monotone function then \(f\) is continuous if and only if \(f(\bigvee D) = \bigvee \{f(x) \mid x \in D\}\)
- for each directed \(D \subseteq X\).
- If \(f : X \rightarrow Y\) is a continuous function then \(f\) is stable if and only if \(\forall x, x' \in X, x \vee x' \in X\) implies \(f(x \wedge x') = f(x) \wedge f(x')\).
- If \(f : X \rightarrow Y\) is a stable function then \(f\) is linear if and only if \(f(\bot) = \bot\) and \(\forall x, x' \in X\) bounded \(f(x \vee x') = f(x) \vee f(x')\).

Definition 15. (STRONGLY STABLE FUNCTION). A function \(f : Q \rightarrow R\) between two qualitative domain with cohesion is strongly stable if for all \(x \in C(Q)\) then \(f(x) \in C(R)\) and \(f(\bigwedge X) = \bigwedge_{a \in X} f(a)\).

The strongly stable functions are similar to stable functions, but they have to preserve coherence as well as intersections of coherent sets of states and not just of bounded ones. The category obtained taking as objects qualitative domains with cohesion and as morphisms the strongly stable functions is cartesian closed.

Definition 16. (PRODUCT). Let \(Q\) and \(R\) two qualitative domain with coherence. Their product \(Q \times R\) is \((Q \& R, C(Q \times R))\).
A set \( X \subseteq A \times B \) is a pairing if \( \pi_1(X) = A \) and \( \pi_2(X) = B \). Recall that the set of strongly stable function between from \( Q \) to \( R \) is a qualitative domain \([16]\).

**Definition 17. (Exponential object).** Let \( Q \) and \( R \) two qualitative domains with coherence. Let \( R^Q \) be the qualitative domain of strongly stable function between from \( Q \) to \( R \). The exponential object \( Q \rightarrow R \) is \( (R^Q, \mathcal{C}(Q \rightarrow R)) \) where, for all \( F \subseteq R^Q \), for all \( X \in \mathcal{C}(Q) \), for all pairing \( P \subseteq X \times F \), \( F \) belongs to \( \mathcal{C}(Q \rightarrow R) \) iff by assuming \( \text{eval}(P) = \{ f(x)(x,f) \in P \} \), the following two conditions hold

- \( \text{eval}(P) \in \mathcal{C}(R) \)
- \( \bigwedge \text{eval}(P) = (\bigwedge F)(\bigwedge X) \)

**Example 2.** Let us consider the qualitative domain \( B \rightarrow B \) of strongly stable function between booleans. Let \( a, b \in \{ \text{tt}, \text{ff} \} \) we define some functions in \( B \rightarrow B \) as follows

\[
\begin{align*}
\text{b}^a &= \begin{cases} 
  b & \text{if } x = a \\
  \bot & \text{otherwise}
\end{cases}
\end{align*}
\]

We noted \( \bot \) the bottom of qualitative domains, corresponding to emptyset in case of coherence spaces.

We sketch the proof that \( F = \{ \text{tt}^u, \text{tt}^u, \text{tt}^\pi \} \) belong to \( \mathcal{C}(B \rightarrow B) \). For all \( X \in \mathcal{C}(B) \) there are two cases.

- If \( \emptyset \not\in X \) then for every pairing \( P \subseteq X \times F \), so we have \( \emptyset \not\in \text{eval}(P) \), since the considered functions are strict. Thus \( \text{eval}(P) \in \mathcal{C}(B) \) (see Example 1).
- If \( \emptyset \in X \) then \( X \) is singleton (see Example 1) and easily we can check that for every pairing \( P \subseteq X \times F \), so we have \( \text{eval}(P) \in \mathcal{C}(B) \).

The second condition of Definition 17, about the commutation of greatest lower bound, can be easily checked to be true.

### 5.2 Second Order Gustave Or

**is not strongly stable**

For sake of simplicity, we prove simply that a boolean version of \( \text{bG} \) is not a strongly stable function. We replace everywhere \( \emptyset \) by \( \text{tt} \) and \( \text{success} \) by \( \text{ff} \), respecting our definitions. Let us consider the second-order Gustave-Or defined in Table 6. It is straightforward to check that the definition of \( \text{bG} \) is well given and it is a boolean generalization of a Second Order Gustave Or.

---

3For sake of conciseness, we use the exponential notation in order to denote pairs of booleans: exponent (base) corresponds respectively to first (second) projection.

We prove that \( \text{bG}_{k_1,k_2,k_3,k_4} \) is not a strongly stable function (see Definition 15) for all \( k_1, k_2, k_3, k_4 \in \{ \text{tt}, \text{ff} \} \). Let us consider the functions in Example 2 above. We take 3 elements of \( (B \rightarrow B) \times (B \rightarrow B) \times (B \rightarrow B) \), namely \( t_1 = (\text{tt}^u, \text{tt}^\pi, \text{tt}^\pi) \), \( t_2 = (\text{tt}^u, \text{tt}^u, \text{tt}^\pi) \) and \( t_3 = (\text{tt}^\pi, \text{tt}^\pi, \text{tt}^\pi) \).

The set \( I = \{ t_1, t_2, t_3 \} \) is in the coherence set of the domain corresponding to \( (B \rightarrow B) \times (B \rightarrow B) \times (B \rightarrow B) \), since the projections are in the coherence set of \( B \rightarrow B \). We have two cases.

- If \( k_1 = k_2 = k_3 \) then \( \text{bG}_{k_1,k_2,k_3,k_4}(I) = k_1 \), but \( \text{bG}_{k_1,k_2,k_3,k_4}(\bigwedge I) = \text{bG}_{k_1,k_2,k_3,k_4}(\emptyset) = \emptyset \).
- Otherwise, there exists \( i, j \in \{ 1, 2, 3 \} \) such that \( k_i \neq k_j \), thus

\[
\text{bG}_{k_1,k_2,k_3,k_4}(I) \notin \mathcal{C}(B)
\]

since

\[
\{ k_1, k_2 \} \subseteq \text{bG}_{k_1,k_2,k_3,k_4}(I).
\]

Therefore \( \text{bG}_{k_1,k_2,k_3,k_4} \) is not strongly stable, for all \( k_1, k_2, k_3, k_4 \).

**Acknowledgments**

We wish to express our gratitude to Antonio Bucciarelli and Marco Gaboardi for some useful discussions about topics considered in this paper.

### 6. REFERENCES


A. Bucciarelli and T. Ehrhard. Sequentiality in an
Computability, M. Davis and E. J. Weuker.
Computability: An Introduction to N. Cutland.
J.-Y. Girard. The system F
M. Gaboardi and L. Paolini. Syntactical, operational
T. Ehrhard. Hypercoherences: A strongly stable
J.-Y. Girard. Linear logic. Theoretical Computer
Proofs and