Linearity and PCF: a semantic insight!

Marco Gaboardi\textsuperscript{1}, Luca Paolini\textsuperscript{2}, and Mauro Piccolo\textsuperscript{3}

\textsuperscript{1}Dipartimento di Scienze dell’Informazione, Università degli studi di Bologna, INRIA Focus Team, Mura Anteo Zamboni 7, 40127 Bologna, Italy
\textsuperscript{2}Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy
\textsuperscript{3}Dipartimento di Elettronica, Politecnico di Torino, Corso Duca Degli Abruzzi 24, 10129 Torino, Italy

Abstract

Linearity is a multi-faceted and ubiquitous notion in the analysis and the development of programming language concepts. We study linearity in a denotational perspective by picking out programs that correspond to linear functions between coherence spaces.

We introduce a language, named $\ell$/PCF\textsuperscript{⋆}, that increases the higher-order expressivity of a linear core of PCF by means of new operators related to exception handling and parallel evaluation. $\ell$/PCF\textsuperscript{⋆} allows us to program all the finite elements of the model and, consequently, it entails a full abstraction result that makes the reasoning on the equivalence between programs simpler.

Denotational linearity provides also crucial information for the operational evaluation of programs. We formalize two evaluation machineries for the language. The first one is an abstract and concise operational semantics designed with the aim of explaining the new operators, and is based on an infinite-branching search of the evaluation space. The second one is more concrete and it prunes such a space, by exploiting the linear assumptions. This can also be regarded as a base for an implementation.

1 Introduction

Linearity is a key tool in order to support a conscious use of resources in programming languages. A non-exhaustive list of its uses includes garbage collection, memory management and aliasing control, description of digital circuits, process channels and messages management, languages for quantum computations, etc. A survey of several variants of linear type systems proposed in literature is [34]. This broad spectrum of applications highlights the fact that linearity is a multifaceted abstract concept which can be considered in different perspectives. For instance, notions of syntactical linearity can be considered when variables are used once (in suitable senses), e.g. [21, 4]. On the other hand, if redexes cannot be discarded or duplicated during reduction [21] then a kind of operational linearity is achieved. This is related to the notion of simple term [24] in $\lambda$-calculus which suggests a kind of linearity on reductions, unrelated from a specific strategy.

Although some ideas that can be tracked to linearity have been implicitly used in programming languages for many years, the introduction of linear logic [19] is a redoubtable milestone in this setting. Linear logic arises from a sharp semantic analysis of stable domains where stable functions have been decomposed into linear functions and exponential domain constructors. Such a decomposition is patently reflected in the syntax of linear logic. Moreover, it suggests a new approach to linearity: denotational linearity. In a programming perspective, denotational linearity says that programs (i.e., closed terms) should correspond via a suitable interpretation to linear function on some specific domains, e.g. the linear models introduced in [22, 19, 13, 14]. By tackling this correspondence minutely some important contributions to the theory of programming can be obtained. For instance, if the intended domain includes all the computable functions, this analysis provides Turing-complete languages with weak syntactic linear constraints on variables, and new linear operators that, in a higher-type computability perspective [25, 27] increase the expressivity of linear languages.

To advance in this research line, we aim to pick out all and only (recursive) linear functions in a linear model built as the full subcategory of coherence spaces and linear functions [19] identified by the type structure of numerals and arrows. The language $\ell$/PCF proposed in [30] is correct for this linear model, so it is denotationally linear. Moreover, it grasps a limited completeness, namely $\ell$/PCF is sufficient to assure the definability of all the tokens (“prime elements” in domain terms) of the linear model. From this follows a restricted full abstraction result for terms without free stable variables (i.e. variables used for recursion). A more general result was erroneously claimed in [30, Corollary 3]; see Section 3 for more details.

In this paper we propose $\ell$/PCF, a language extending $\ell$/PCF by the operator $\ell$et-$\ell$or. This operator provides a linear counterpart of the gor operator introduced in [29], and increases the non-deterministic expressiveness of the language. Indeed,
this extension allows us to gain a finite definability result [15], i.e. the definability of all the finite cliques of the considered linear model. In fact, a crucial role in the proof of our finite definability result is played not only by the let-for operator but also by the $\ell$et-for operator which. In [30], the which? operator has been proposed as an example of interesting higher-order linear operator providing run-time information, in order to give a flavor of our research line. However, it was not necessary for the token-definability and the limited full abstraction results. Here, we show how to use which? (together with the new let-for operator) in order to reach the finite definability.

By using the finite definability, we prove that $\ell$PCF, is fully abstract with respect to the considered linear model. That is, the operational equivalence coincides with the denotational equivalence. This result allows us to reason on programs in a compositional way. This is important because even if the operational equivalence is defined for closed ground contexts, since it relies on the operational formal machine evaluating programs, sometimes one wants just to replace a subterm with another one and preserves the equivalence. So, we need to consider also open sub-terms that form our programs. In our case, tackling the equivalence also when terms contain open variables is particularly challenging due to the presence of stable variables, used for recursion. Besides, we prove the coincidence of three different definitions of operational equivalence that make simpler the reasoning on the equivalence between programs by permitting to consider only contexts of a restricted shape. Moreover, the proof of this result uses non-trivial syntactical and denotational arguments that are to our knowledge new and of wider interest.

We remark that $\ell$PCF is neither syntactically linear nor operationally linear in a tight sense, albeit its finitary fragment (the set of programs which does not involve recursion) enjoys some syntactical and operational forms of linearity. For instance, it is syntactically linear for slices, and when only slices are considered it can be evaluated without duplicating redexes. Besides, we conclude the paper by giving an operational semantics inducing an efficient evaluation of $\ell$PCF terms. This operational semantics traces out and records linear information to drastically prune the infinite branching search tree of the evaluation of $\ell$PCF.

Outline  In Section 2 we introduce in an informal way the contributions that will be technically presented in the rest of the paper. In Section 3 we introduce the background needed to understand the technical results of this paper. In Section 4 we show that $\ell$PCF lacks the full abstraction. In Section 5 we introduce the language $\ell$PCF, and we give some programming examples. In Section 6 we prove the finite definability and the full abstraction for $\ell$PCF, In Section 7 we show the coincidence of the three operational equivalence introduced. In Section 8 we give an abstract machine for $\ell$PCF, that traces the linear use of terms and it provides the base for an efficient implementation of our language.

2  Contributions: An Informal Account

In this paper, we propose an extension of the language $\ell$PCF introduced in [30]. $\ell$PCF is a PCF-like language enriched with the which? operator, that is correct for the considered linear model. In particular, linearity is obtained by means of some constraints on clever variable management. $\ell$PCF is based on three kinds of variables: ground $x^\ast$ and stable variables $f^{{\sigma} \rightsquigarrow {\tau}}$, that can be weakened and contracted, and linear ones $f^{{\sigma} \rightsquigarrow {\tau}}$ that cannot. Ground and linear variables can be $\lambda$-abstracted, stable variables cannot. Stable variables can be bound by a dedicated binder (the $\mu$-abstraction) and they are used to define recursive functions. In $\ell$PCF, an argument $n$ supplied to $\lambda x. n$ is evaluated by using a call-by-value policy in case $x$ is ground, a call-by-name policy otherwise. More details can be found in Section 3.2.

The denotational insights

The basic components of the model are tokens. A token is a tuple of natural numbers that complies with the structure of types. For instance

$$\begin{align*}
((\lambda \tau. \iota) \rightsquigarrow (\iota \rightsquigarrow (\iota \rightsquigarrow (\iota \rightsquigarrow 2)) \rightsquigarrow (3 \rightsquigarrow 4)) \rightsquigarrow 5)
\end{align*}$$

The second row of the formula above describes a token belonging to the coherence space corresponding to the type written in the first row (clearly, $\iota$ is the type of natural numbers while $\to$ is the linear arrow). $\ell$PCF is able to define all the tokens which belong to coherence spaces corresponding to the types [30]. For instance, the following term defines the token written above:

$$\begin{align*}
\lambda f. \lambda g. \ell \text{if} \left( (f(\lambda x. \ell \text{if} (x \cong 0) 1 0) \cong 2) \text{ and (g 3 \cong 4)} \right) 5 0.
\end{align*}$$

To program a token means to verify that each input coincides with the one described by the token, in the example there are two input (i.e. $((0,1),2)$ and $(3,4)$).

The elements of the linear model as usual in coherence spaces are cliques, i.e. sets of coherent tokens. The coherence relation ensures that cliques describes only traces of functions (where the trace is an economic way to describe the graph of a function).
In particular, coherence establishes when two tokens can coexists in the trace of a function. Consider a simple clique of a linear function:

\[
\{ ((0,0) , ((5,7), 0) , 0 ) \\
((2,0) , ((3,9), 1) , 1 ) \\
((2,1) , ((3,9), 1) , 2 )
\]

Coherence ensures that, for each pair of tokens, there exists an input argument making us able to decide what is the unique token in the considered pair being (eventually) involved in the computation. For instance, by applying the first argument to \(\lambda\) we can distinguish between the second and the third token, in fact the sub-tokens \((2,0)\) and \((2,1)\) are incoherent indeed they cannot coexist in the same function. Likewise, by applying the second argument to \(\lambda^x.\ellif (x \doteq 5) ? (\ellif (x \doteq 3) \varnothing \varnothing)\), we can distinguish the first token from both the second and the third, indeed \(((3,9),1)\) and \(((5,7),0)\) are incoherent and they cannot coexist in the same function.

These properties are important in order to program linear functions, however, linearity gives no information about how to locate such observations. In particular, when higher-order types are considered this becomes quite tricky, and parallelism is necessary.

Is \(\lambda\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lor\lo
with the provision that the basis \( \Gamma \) does not contain linear variables. This operator can then be equipped by the following semantics:

\[
\begin{align*}
M_1 \Downarrow \circ \quad & M_2 \Downarrow \circ \quad n + 1 \\
\text{Gor}(M_1, M_2, M_3) \Downarrow \circ \quad & n \\
M_2 \Downarrow \circ \quad & M_3 \Downarrow \circ \quad n + 1 \\
\text{Gor}(M_1, M_2, M_3) \Downarrow \circ \quad & n \\
M_1 \Downarrow \circ \quad & M_1 \Downarrow \circ \quad n + 1 \\
\text{Gor}(M_1, M_2, M_3) \Downarrow \circ \quad & n 
\end{align*}
\]

Adding it to S/PCF however is not sufficient to program the clique defined in Equation 1 (actually, to extend S/PCF by \text{Gor} does not add any linear function!). We need something that permits to linear variable to permit parallel observations of the same linear variable on different arguments. One way to do this is by introducing a further control operator, that can be described as a \text{let}\-like operator:

\[
\begin{align*}
\Gamma \vdash N : \sigma \rightarrow \tau, \quad F : \sigma \rightarrow \tau \Delta \vdash M : \iota \\
\Gamma; \Delta \vdash \text{let}\ f^{\sigma \rightarrow \tau} = N \text{ in } M : \iota
\end{align*}
\]

where the variable \( F \) can be weakened and contracted, i.e. it does not respect any occurrence constraint. The operational meaning of this \text{let}\-like operator can be described by the following rule:

\[
[\Pi] = \{ a \} \subseteq [N] \quad M[\Pi/F] \Downarrow \circ \quad n \\
M[\Omega/F] \Uparrow\quad \text{let}\ f^{\sigma \rightarrow \tau} = N \text{ in } M \Downarrow \circ \quad n
\]

Such an operator appears to have the flavor of the co-dereliction of Differential Linear Logic [16]. The idea motivating the above rule is that:

\textit{Denotational linearity} does not conflict with several observations of a \text{clique} performed on the \textit{same token}.

Following this intuition one can read the side condition \( [\Pi] = \{ a \} \subseteq [N] \) as saying that \( \Pi \) is interpreted as the unique token of \( N \) that can be so observed several time in evaluating \( M[\Pi/F] \). The third condition \( \Pi[\Pi/F] \Uparrow\) instead is here just to ensure that the token is actually really used.

A clever combination of the two above operators allows to program the clique described in Equation 1 (such combination will be detailed in Section 5.1). We remark some drawbacks of \text{let}.

1. Its operational rule given above is not effective, due to the presence of the third condition \( M[\Omega/F] \Uparrow\).
2. It limits the understanding of the program control flow. It can be checked only at run-time that the same token has been observed.

The first drawback can be solved by designing an ad-hoc operational semantics, the second one is more problematic since it could make the understanding of programs really problematic. For these two reasons, we extend S/PCF by a single operator, combining together \text{Gor} and \text{let}. The \text{let}\-\text{forc} (considering only a linear variable \( f \)) is typed as follows:

\[
\begin{align*}
\Gamma \vdash N : \sigma \quad f : \sigma, \Delta \vdash M_1 : \iota \\
\Delta \vdash M_2 : \iota \\
\Delta \vdash M_3 : \iota \\
\Gamma; \Delta \vdash \text{let}\ f^{\sigma \rightarrow \tau} = N \text{ in } M_1 M_2 M_3 : \iota
\end{align*}
\]

again with the provision that the basis \( \Delta \) does not contain linear variables. Note that now the \text{let}\-\text{forc}-bounded variable \( f \) is linear, albeit used in three program-branches. The \text{let}\-\text{forc} evaluation can now be described using rules as follows:

\[
\begin{align*}
M_1[\Pi/f] \Downarrow \circ \quad & M_2[\Pi/f] \Downarrow \circ \quad n + 1 \\
\text{let}\ f^{\sigma \rightarrow \tau} = N \text{ in } M_1 M_2 M_3 \Downarrow \circ \quad & n
\end{align*}
\]

\( (we\ give\ just\ one\ rule,\ the\ other\ two\ are\ analogous)\). Thanks to a generalized \text{let}\-\text{forc} operator, the resulting language S/PCF, permits to program all the finite cliques. So, the \textit{full abstraction} follows.

\textbf{The operational Insights}

The full abstraction result mentioned above ensures as usual that a \textit{compositional} theory of program equivalence can be defined. That is, program equivalence is a congruence. Concretely, the full abstraction is proved with respect to a non-standard notion of contextual equivalence \( \sim \), named \textit{fix-point equivalence} defined as:

\[
M \sim N \iff C[M[P^\delta/F^\gamma]] \Downarrow \circ \quad n \iff C[N[P^\delta/F^\gamma]] \Downarrow \circ \quad n
\]

where all the \( P^\sigma_i \) are closed terms. This notion of equivalence makes explicit the fact that for reasoning in a compositional way about programs we need to permit to substitute general terms to stable variables, even if such variables are only used for dealing
with recursion. It is exactly to program the \( P \), that the definability of all the finite cliques is needed in order to get full abstraction. Indeed, in [30] a full abstraction results for terms without open stable variables was proved. Anyway, the definition of fix-point equivalence seems ad-hoc with respect to the usual notion of contextual equivalence:

\[
M \simeq N \iff C[M] \Downarrow n \iff C[N] \Downarrow n
\]

So, it is natural to consider the following question:

Do the relation \( \sim \) and \( \approx \) coincide?

Anticipating, the answer to this question is positive. Although, proving this result is quite technical. Intuitively, the reason is that stable variables can only be \( \mu \)-abstracted. More precisely, suppose \( M \neq N \) and suppose they have a free stable variable \( f \) of type \( \sigma_1 \to \ldots \to \sigma_k \to i \). We have that there is a context \( C \) and a term \( P \) such that \( C[M[P/\!f \!]] \Downarrow n \) and \( C[N[P/\!f \!]] \not\Downarrow n \). Intuitively, to prove that also \( M \neq N \) holds, one could think to build a context

\[
C' = C[(\lambda f. \ldots)[P]]
\]

such that \( C'[M] \Downarrow n \) and \( C'[N] \not\Downarrow n \). Unfortunately, this context cannot be built since the stable variable \( f \) cannot be \( \lambda \)-abstracted. So, the best that one can hope to get is a context

\[
C'' = C_1[\mu f. C_2[(\ldots)]]
\]

acting similarly to \( C' \). Yet linearity comes in our help. We will show in Section 7 that this kind of contexts always exist. We propose an example, suppose \( C \) be the empty context \([\ldots]\), and suppose that \( M \) and \( N \) can be distinguished by using \( P \) (for simplicity, we assume \( P \) typed \( \ell \to i \) like \( f \), and \([P]\) = \{(0, m)\}). So, by considering a term \( P' \) defining the clique \{(0, 0), (n, m)\} we can define \( C'' \) as

\[
\mu f. \lambda y. (\lambda x. \ell i f (x = 0) \ldots)[\ldots](P' y)
\]

This context can then be used to build the terms \( C''[N] \) and \( C''[N] \). After one recursion step, the terms \( N[C''[N]/f] \) and \( M[C''[M]/f] \) are built. However, in the next recursive step \( C''[N] \) and \( C''[N] \) will behave exactly as \( P \) and so the two terms can be distinguished. One can doubt that this construction cannot be always done. However, the properties of the linear model ensure that finite cliques are never maximal with respect to the set theoretical inclusion. This for instance does not happen in stable models in general. This means that given a finite clique \( x \) one can always find a new coherent token that can be added to it and that can be used to control the recursion. The proof just generalizes this example.

Besides the equivalence of programs, linearity provides also crucial information for the operational evaluation of programs. As instance, the side condition \([P] = \{a\} \subseteq [N]\) in the rule of \( \ell \text{-for} \) above assumes the existence of such a \( P \), but it does not give any hints on its search. If we face a concrete implementation of \( \mathcal{S}/\text{PCF} \), then this could be a problem. An exhaustive search of such a token \( a \) is intrinsically inefficient. So, it is natural to consider the following question:

Is there a reasonable way to drive the evaluation of \( \mathcal{S}/\text{PCF} \) programs?

The answer to this question as we will show in Section 8 is positive, and again linearity comes in our help. It ensures two important properties: first that such a token \( a \) exists, and second that it is unique. From these properties we can devise a finer implementation of the language. Consider again the rule

\[
\frac{M_1[P/f] \Downarrow n \quad M_2[P/f] \Downarrow n + 1 \quad [P] = \{a\} \subseteq [N]}{\let f = N \in \text{for } M_1 M_2 \Downarrow n}
\]

Instead of evaluating \( M_1[P/f] \) one can think to evaluate \( M_1 \) in an environment \( e \) storing the information about variables. So, we associate \( f \) to the term \( N \). When the term \( N \) is used, we record its observed token \( a \) (conveniently encoded by a term \( P \)). In particular, the above \( \ell \text{-for} \) rule becomes, roughly:

\[
\frac{\langle M_1[e_0[f := N]\downarrow] \mid Q[e_1[f := P]\downarrow] \mid M_2[e_2[f := P]\downarrow] \mid (m, e_2) \downarrow (m, e_2) \mid (\let f = N \in \text{for } M_1 M_2 e_0) \downarrow (m, e_2)}{(\let f = N \in \text{for } M_1 M_2 e_0) \downarrow (m, e_2)}
\]

Note that the above reasoning relies on an effective tracing of all evaluated terms. For instance an evaluation of the term \( (\lambda x'. M)Q \) is done using a rule as

\[
\frac{\langle Q[e_0]\downarrow (m, e_1) \mid (M[e_1[x := m]\downarrow] \mid (n, e_2) \downarrow (n, e_2) \mid (\ell Q[e_0[f := \lambda x' M^\prime]\downarrow] (m, e_2[f := (m, n)])) \downarrow (m, e_2[f := (m, n)])}}{(\ell Q[e_0[f := \lambda x' M^\prime]\downarrow] (m, e_2[f := (m, n)])) \downarrow (m, e_2[f := (m, n)])}
\]

that traces the information of the used token, i.e. it puts \((m, n)\) into the environment. A tracing evaluation of \( \mathcal{S}/\text{PCF} \) programs will be devised in Section 8. Note that this kind of evaluation can be considered a kind of linear call-by-need evaluation where only one evaluation is done and where further observations are used to check the consistency of the information.
Synopsis

In summary, the key contributions of this paper are:

- The definition of a new linear operator \( \ell \cdot \ell \times \) (Definition 13) that permits to establish a bridge between denotational and syntactical linearity through a full abstraction result (Corollary 2).
- A finite definability result (Theorem 6). It gives evidence that \( S/PCF \) is able to program a broader class of linear programs than other linear programming languages.
- The coincidence of different operational equivalences (Corollary 5). This makes simpler the reasoning on the equivalence between linear programs.
- An efficient reduction semantics exploiting linear properties in order to provide a concrete running evaluation of \( S/PCF \), that avoids exhaustive evaluation searches, by tracing and recording explicitly the linear use of subterms.

3 Background

3.1 Coherence Spaces

Coherence spaces are a simple framework for Berry’s stable functions [7], developed by Girard [19]. More details are in [20].

A coherence space \( X \) is a pair \( (|X|, \succeq_X) \) where \(|X|\) is a set of tokens called the web of \( X \) and \( \succeq_X \) is a reflexive and symmetric relation between tokens of \(|X|\) called the coherence relation on \( X \). The strict incoherence \( \prec_X \) is the complementary relation of \( \succeq_X \); the incoherence \( \asymp_X \) is the union of relations \( \prec_X \) and \( = \); the strict coherence \( \dashv_X \) is the complementary relation of \( \asymp_X \). A clique \( x \) of \( X \) is a subset of \(|X|\) containing pairwise coherent tokens. The set of cliques of \( X \) is denoted \( \mathcal{C}(X) \), while the set of finite cliques is denoted \( \mathcal{C}_{f in}(X) \).

The basis of our model is the infinite flat domain. Let \( N \) denotes the space of natural numbers, namely \((|N|, \succeq_N)\) such that \(|N| = \mathbb{N}\) and \( m \preceq_N n \) if and only if \( m = n \), for all \( m, n \in |N| \).

Definition 1. Let \( X \) and \( Y \) be coherence spaces and \( f : \mathcal{C}(X) \rightarrow \mathcal{C}(Y) \) be a monotone function. Then, \( f \) is linear whenever \( \forall x \in \mathcal{C}(X), \forall b \in f(x) \exists ! a \in x \) s.t. \( b \in f(\{a\}) \).

Linear functions can be represented as cliques.

Definition 2. Let \( X \) and \( Y \) be coherence spaces. \( X \rightarrow Y \) is the coherence space having \(|X| \rightarrow |Y|\) as web, while \((a, b) \succeq_{X \rightarrow Y} (a', b') \) iff \( a \succeq_X a' \) implies \( b \succeq_Y b' \).

The trace of a linear function \( f : \mathcal{C}(X) \rightarrow \mathcal{C}(Y) \) is \( \mathcal{T}(f) = \{ (a, b) \mid a \in |X|, b \in f(\{a\}) \} \). Given \( t \in \mathcal{C}(X \rightarrow Y) \) and \( x \in \mathcal{C}(X) \), let us define the map \( \mathcal{F}(t) : \mathcal{C}(X) \rightarrow \mathcal{C}(Y) \) as

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

Lemma 1. If \( f : \mathcal{C}(X) \rightarrow \mathcal{C}(Y) \) is a linear function then \( \mathcal{T}(f) \in \mathcal{C}(X \rightarrow Y) \). If \( t \in \mathcal{C}(X \rightarrow Y) \) then \( \mathcal{F}(t) : \mathcal{C}(X) \rightarrow \mathcal{C}(Y) \) is a linear function.

Definition 3 (Linear Model). The Linear Model \( \mathcal{L} \) is the type structure generated by the coherence space \( N \) and the arrow \( \rightarrow \).

3.2 The language \( S/PCF \)

\( S/PCF \) has been introduced in [17, 30] to be the syntactical counterpart of the above mentioned linear model.

Definition 4. The set \( \mathbb{T} \) of linear types is defined by the grammar:

\[
\sigma, \tau ::= \iota \mid \sigma \rightarrow \tau \quad \text{where } \iota \text{ is the only ground type.}
\]

For the sake of clearness we introduce three kinds of variables, in order to remark their different explicit use.

Definition 5. Let \( \mathbb{V}_\sigma \) be numerable sets of variables of type \( \sigma \). Let \( \mathcal{S}\mathbb{V}_\sigma \) (\( \sigma \neq \iota \)) be numerable sets of variables disjoint from \( \mathbb{V}_\sigma \). Variables in \( \mathbb{V}_\sigma \) are named ground variables. Variables in \( \mathcal{S}\mathbb{V} \) are named linear variables. Variables in \( \mathcal{S}\mathbb{V} = \bigcup_{\sigma \neq \iota} \mathcal{S}\mathbb{V}_\sigma \) are named stable variables.

Note that there are no stable variables of type \( \iota \). Latin letters \( x^\sigma, y^\sigma, f^\sigma, \ldots \) denote variables in \( \mathbb{V}_\sigma \). We use types to explicitly distinguish between ground, e.g. \( \pi \), and linear variables, e.g. \( x^\sigma \rightarrow \tau \). Moreover, \( \iota, F_{\sigma_1}, f_{\sigma_2}, f_{\sigma_3}, \ldots \) denote stable variables.

Last, \( \iota \) is a wild-card for all the variables.

We define terms (Definition 7) as the pre-terms (Definition 6) that can be typed using the type system in Table 1.a.
\[\vdash 0 : \ell (z)\]
\[\vdash t : \ell \to \ell (s)\]
\[\vdash p : \ell \to \ell (p)\]
\[\vdash \text{which?} : ((\ell \to \ell) \to \ell) \to \ell (w)\]
\[\Gamma \vdash M : \tau (w)\]
\[\Gamma, x_1 : \tau, x_2 : \tau \vdash M : \tau (g)\]
\[\Gamma, F, \ell, F' \vdash M : \tau (sw)\]
\[\Gamma, F, F' \vdash M[\ell / F, \ell' / F'] : \tau (sc)\]
\[\Gamma, \Delta \vdash M : \ell (l)\]
\[\Gamma, \Delta \vdash \ell \text{if } M \ell R : \ell (lif)\]
\[\Gamma, \Delta \vdash M : \ell (a)\]
\[\Gamma, \Delta \vdash M : \ell (a)\]
\[\Gamma, \Delta \vdash M : \ell (a)\]
\[\Gamma, \Delta \vdash MN : \tau (ap)\]

---

Table 1: (a) Type system, (b) operational semantics and (c) linear interpretation for \( \llcorner \text{PCF} \lrcorner \)

**Definition 6.** Pre-terms are defined by the grammar:

\[
\begin{align*}
M &::= \lambda x^\sigma \, t | 0 | s | p | \ell \text{if } M \text{ M M} | (M) | (\lambda x^\sigma . M) | \mu F . M | \text{ which?}
\end{align*}
\]

We write \( \mathbb{N} \) for \( s(\cdots(s(0)\cdots) \) where \( s \) is applied \( n \)-times to 0, and we denote \( \mathbb{N}^* = \{0, \ldots, \mathbb{N}, \ldots\} \) the set of numerals.

We consider typing judgments of the shape \( \Gamma \vdash M : \sigma \) where \( \Gamma \) is a pre-term, \( \sigma \) is a linear type and \( \Gamma \) is a basis, that is a finite list of variables in \( \text{Var} \), where each variable appears at most once. We denote \( \Gamma|\Delta|\ell \) the restriction of the basis \( \Gamma \) containing only variables in \( \text{SVar} \) (resp. \( \text{VVar}^r, \Delta \text{Var} \)). We denote \( \Gamma, \Delta \) and \( \Gamma \cap \Delta \) the union and the intersection of two basis respectively. We can now define \( \llcorner \text{PCF} \lrcorner \) terms.

**Definition 7.** The terms of \( \llcorner \text{PCF} \lrcorner \) are the pre-terms typable by using the type system in Table 1.a.

Sometimes, we write \( \mathcal{M}^\sigma \) when, for some \( \Gamma \), we have \( \Gamma \vdash M : \sigma \). Free variables of any kind (FV), free linear variables (LFV), free stable variables (SVF), closed and open terms are defined as expected. We denote \( \mathcal{P} = \{\mathcal{M}^\sigma \in \llcorner \text{PCF} \lrcorner \mid \text{FV}(\mathcal{M}^\sigma) = \emptyset\} \) the set of programs. As usual, \( \mathcal{M}[N/\sigma] \) denotes the capture-free substitution of all free occurrences of \( \sigma \) in \( \mathcal{M} \) by \( N \).

**Lemma 2** (Substitution). Let \( \mathcal{M}^\sigma, \mathcal{N}^\tau \in \llcorner \text{PCF} \lrcorner \).
- If \( \ell \text{FV}(\mathcal{M}^\sigma) \cap \ell \text{FV}(\mathcal{N}^\tau) = \emptyset \) and \( x^\sigma \in \text{VVar} \) then \( \mathcal{M}[\mathcal{N}^\tau/x^\sigma] \in \llcorner \text{PCF} \lrcorner \).
- If \( \ell \text{FV}(\mathcal{N}^\tau) = \emptyset \) then \( \mathcal{M}[\mathcal{N}^\tau/f^\sigma] \in \llcorner \text{PCF} \lrcorner \).
- If \( \ell \text{FV}(\mathcal{N}^\tau) = \emptyset \) then \( \mathcal{M}[N/x_1, \ldots, N/x_n] \in \llcorner \text{PCF} \lrcorner \).

Pairing (i.e. mapping pair of numerals on one numeral) and projections function will be used everywhere in the paper. We will denote with \( [\mathbb{N}, \mathbb{N}] \) the numeral \( k \) encoding the ordered pair of \( \mathbb{N} \) and \( \mathbb{N} \) and we write \( \pi_1(k) \) for the numeral \( a_1 \) and \( \pi_2(k) \) for the numeral \( a_2 \). \( \llcorner \text{PCF} \lrcorner \)-terms defining them can be found in [30]. Note that we avoid the use of an explicit product data type in order to adhere as much as possible to the modeled type structure.

**Definition 8.** The evaluation relation \( \downarrow \subseteq \mathcal{P} \times \mathcal{N} \) is the smallest relation inductively satisfying the rules of Table 1.b. If there exists a numeral \( \mathbb{N} \) such that \( \mathcal{M} \downarrow \mathbb{N} \) then we say that \( \mathcal{M} \) converges, and we write \( \mathcal{M} \downarrow \), otherwise we say that it diverges, and we write \( \mathcal{M} \uparrow \).

Remark that \( p \) is a partial operator, namely \( p \, 0 \) diverges. The set of \( \sigma \)-context \( \text{Ctxt}_\sigma \) is defined as:

\[
\begin{align*}
\text{C}_\sigma &::= [\sigma] | x^\sigma | 0 | s | p | \ell \text{if } C_\sigma | C_\sigma | C_\sigma | C_\sigma \mid \text{ which?} | (C_\sigma[C_\sigma]) | (\lambda x^\sigma . C_\sigma) | \mu F . C_\sigma
\end{align*}
\]
\(C[N^\tau]\) denotes the result obtained by replacing all the occurrences of \(\sigma\) in the context \(C[\sigma]\) by the term \(N^\tau\) and by allowing the capture of its free variables.

**Definition 9** (Standard Operational Equivalence). Let \(M^\sigma, N^\tau \in \text{S\textsc{PCF}}\).

- \(M \lessdot_{\sigma} N\) whenever, for all \(C[\sigma]\) s.t. \(C[M], C[N] \in \mathcal{I}\), if \(C[M] \not\lessdot_{\sigma} \emptyset\) then \(C[N] \not\lessdot_{\sigma} \emptyset\).
- \(M \approx_{\sigma} N\) if and only if \(M \lessdot_{\sigma} N\) and \(N \lessdot_{\sigma} M\).

We are interested in a language for which the linear model \(L\) is fully abstract under a standard interpretation \([-\cdot]\), i.e., ground types are interpreted on flat posets (see [33]). The standard interpretation is such that \([x] = N\) and \(\sigma : \tau \rightarrow \emptyset = [\sigma] \rightarrow [\tau]\).

An environment \(\rho \in \text{Env}\) is a partial function mapping a variable \(x^\sigma\) in a token \(a \in [[\sigma]]\) and a stable variable \(F^\sigma\) in a finite clique \(x \in Cl_{fin}([[\sigma]])\). The set of environments is denoted by \(\text{Env}\). Let \(\ell\) be a sequence of tokens of a coherence space, let \(x\) be a sequence of non-stable variables of the same length of \(\ell\); \(\ell\) is the environment such that \(\rho[x] := \ell(i) = a_i\) in case \(x^i\) is the \(i\)-th element of \(x\), otherwise \(\rho[x] := \ell(i)(x^i) = \rho(x^i)\). If \(\ell\) is a sequence of finite cliques and \(F^\sigma\) is a sequence of stable variables of the same length then \(\rho[F^\sigma := \ell]\) is defined likewise.

**Definition 10.** Let \(M^\sigma, N^\tau \in \text{S\textsc{PCF}}\) and \(\rho \in \text{Env}\). The linear interpretation \([[M^\sigma]] : \text{Env} \rightarrow Cl([[\sigma]])\) is defined in Table 1.c using \(\mathcal{F}\) as defined in Equation 2 and \(\mathcal{F}\) which is the least fix point operator.

### 3.3 Stable Closed Full Abstraction

We recall the main properties of \(\text{S\textsc{PCF}}\) [30, 32]. First, the linear interpretation is adequate and correct.

**Theorem 1** ([30]). Let \(M^\sigma \in \mathcal{I}\) and \(N^\tau, L^\sigma \in \text{S\textsc{PCF}}\).

- Adequacy: \(M^\sigma \not\lessdot_{\sigma} N\) \iff \([[M^\sigma]] = [[N]]\).
- Correctness: \(N^\tau = [[L^\sigma]] \Rightarrow N \approx_{\sigma} L\)

The proof of this theorem can be found in Appendix A.1.

In \(\text{S\textsc{PCF}}\) all the tokens of the linear model \(L\) can be defined. Moreover, as shown in [30], this can be done without using the \(\text{which}\?)\ operator.

**Theorem 2** (Token Definability). If \(u \in [[\sigma]]\) then there exists a closed \(M^\sigma \in \text{S\textsc{PCF}}\) such that \([[M]] = \{u\}\).

**Proof.** If \(\sigma \in \mathcal{T}\) then, as usual, its order ORDER(\(\sigma\)) is defined by ORDER(\(i\)) = 0 and ORDER(\(\sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow \emptyset\)) = 1 + \(\max\{\text{ORDER}(\sigma_i)\} i \in [1, k]\). The proof is done by induction on the order of \(\sigma\). The case ORDER(\(\sigma\)) = 0 is trivial.

Assume ORDER(\(\sigma\)) = 1, thus \(\sigma = \tau_1 \rightarrow \ldots \rightarrow \tau_k \rightarrow \emptyset\). 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

\[
\begin{align*}
M^\sigma &= \lambda x_1^\tau \ldots x_k^\tau . f^\tau \; (x_1^\tau = n_1) \text{ and } (x_2 = n_2) \text{ and } \ldots \text{ and } (x_k = n_k) \; \not\lessdot_{\sigma} \Omega^\tau
\end{align*}
\]

then \([[M^\sigma]]_\rho = \{a\}\) for all \(\rho\). Finally, assume \(\sigma = \sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow \emptyset\) where ORDER(\(\sigma_i\)) \(\geq 1\) and \(\sigma_i = \tau_i^1 \rightarrow \ldots \rightarrow \tau_i^j \rightarrow \emptyset\) for \(1 \leq i \leq k\). If \(a \in [[\sigma]]\) then \(a\) has shape \(((a_1^1, \ldots, a_{h_1}^1, n_1), \ldots, (a_i^1, \ldots, a_{h_i}^1, n_k), n')\) where \(n_1, \ldots, n_k, n' \in \mathbb{N}\) and \(a_i^1 \in [[\tau_i^1]]\), for each \(i \in [1, k]\) and \(j \in [1, h_i]\). By inductive hypothesis’s, for all \(i \in [1, k]\) and \(j \in [1, h_i]\), there exists a closed term \(M_i^{a_i^1}\) such that \([[M_i^{a_i^1}]] = \{a_i^1\}\). Thus,

\[
\lambda x_1^{\tau_1^1} \ldots x_k^{\tau_k^1}. f^{\tau_k^1} \; (x_1^{\tau_1^1} = n_1) \text{ and } \ldots \text{ and } (x_k^{\tau_k^1} = n_k)\]

is the the closed term defining the considered token.

Token definability permits to define the separating terms used in the next lemma. This in contrast to what happens in [33, 29] where a finite definability is needed.

**Lemma 3** (Separability). Let \(\sigma \in \mathcal{T}\). For all distinct \(f, g \in Cl([[\sigma]])\) there exists a closed term \(P^{\sigma \rightarrow \sigma}\) such that \(\mathcal{F}(\mathcal{P})(f) \neq \mathcal{F}(\mathcal{P})(g)\).
Proof. Let \( \sigma = \sigma_1 \ldots \sigma_k \to \iota \). Since \( f \neq g \) there exists \( a \) such that \( a \in f \) but \( a \not\in g \). Assume \( a = (a_1, \ldots, a_k, n) \) (where \( a_1 \in [\sigma_1], \ldots, a_k \in [\sigma_k], n \in \mathbb{N} \)). By Theorem 2, for each \( i \in [1, k] \) there is \( \mathcal{N}^\sigma_i \in \text{S/PCF} \) such that \( [\mathcal{N}^\sigma_i] = a_i \). By choosing \( \mathcal{M}^\sigma \to f \lambda \iota. f x \) \( (\ x \mathcal{N}^\sigma_i \ldots \mathcal{N}^\sigma_k) = n \) \( \uparrow \uparrow \), the proof follows. \( \square \) \( \square \)

From this follows that full abstraction holds for all the terms that do not contain free stable variables. We stress that such a result was erroneously claimed in [30] to be of more generality. Indeed, we show in the next section that the unrestricted full abstraction fails.

**Theorem 3** (Stable Closed Completeness). Let \( \mathcal{M}^\sigma, \mathcal{N}^\sigma \in \text{S/PCF}^\sigma \) and \( \text{SFV}(\mathcal{M}^\sigma) = \text{SFV}(\mathcal{N}^\sigma) = \emptyset \). Then:

\[
\mathcal{M} \approx^\sigma \mathcal{N} \Rightarrow [\mathcal{M}] = [\mathcal{N}]
\]

**Proof.** Let us prove the contrapposition: let us assume \([\mathcal{M}] \rho \neq [\mathcal{N}] \rho \), for an environment \( \rho \). By Separability Lemma 3, there exists a closed \( \mathcal{P}^\sigma \to f \) such that \( \mathcal{F}([\mathcal{P}] \rho) ([\mathcal{M}] \rho) = n_1 \neq \mathcal{F}([\mathcal{P}] \rho) ([\mathcal{N}] \rho) \). Moreover, by Token Definability Theorem 2 we can build a context \( \mathcal{C}^\sigma \) such that \([\mathcal{P} \mathcal{M}] \rho = [\mathcal{C} \mathcal{P} \mathcal{M}] \emptyset \) and \([\mathcal{P} \mathcal{N}] \rho = [\mathcal{C} \mathcal{P} \mathcal{N}] \emptyset \) where \( \emptyset \) is the empty environment. So, by adequacy we have \( [\mathcal{C} \mathcal{P} \mathcal{M}] \rho \not\approx [\mathcal{C} \mathcal{P} \mathcal{N}] \emptyset \). So \( \mathcal{M} \not\approx^\sigma \mathcal{N} \). \( \square \)

Note that, in the above proof the Token Definability Theorem 2 allows us to build the context \( \mathcal{C}^\sigma \) only because we have assumed that the two terms have no stable variables. In the next section, we prove that in order to relax this constraint the language need to be extended.

**Corollary 1** (Stable Closed Full Abstraction). Let \( \mathcal{M}^\sigma, \mathcal{N}^\sigma \in \text{S/PCF}^\sigma \) and \( \text{SFV}(\mathcal{M}^\sigma) = \text{SFV}(\mathcal{N}^\sigma) = \emptyset \). Then:

\[
\mathcal{M} \approx^\sigma \mathcal{N} \iff [\mathcal{M}] = [\mathcal{N}]
\]

**4 Lack of Full abstraction**

\( \text{S/PCF} \) does not enjoy the unrestricted full abstraction. In this section, we show that the Corollary 1 cannot be extended to terms having free occurrences of stable variables, since they are interpreted in finite cliques. We prove that \( \text{S/PCF} \) is not able to define all finite cliques of the model by presenting a linear function that is not definable in \( \text{S/PCF} \), since it is not strongly stable [12]. We use this function to define two terms having free occurrences of stable variables which are operationally equivalent, albeit they are interpreted into two different linear functions.

**4.1 Token enumeration**

First, we introduce some abbreviations that are useful in order to simplify the rest of the paper. We use \( \langle \text{M and N} \rangle \) to abbreviate \( (\text{if } \text{M} (\text{if } \text{N} \ 0 \ 1) \ 1) \). The equivalence among numerals, denoted \( \equiv \) used in fixed notation, is encoded as \( \mu^f = (\sigma \to \iota^f, \lambda \iota^f. \mu^f x \ (\text{if } \text{y } 0 \ 1 \ (\text{if } \text{z } 1 \ (\text{if } \text{p } \text{x } \text{y}))))) \). Moreover, we define a family of diverging terms by induction on types, \( \Omega^i = \text{p}_0 \) and, if \( \sigma_0 = \mu_1 \ldots \mu_n \to \iota \) for some \( m \in \mathbb{N} \) then \( \Omega^{\sigma_0 \ldots \sigma_{n-1}} = \lambda x_0^{\sigma_0} \ldots x_n^{\sigma_n}. \text{if} (\Omega^1 x_1^{\sigma_1} \ldots x_n^{\sigma_n}) (x_0 \Omega^{\mu_1} \ldots \Omega^{\mu_n}) (x_0 \Omega^{\mu_1} \ldots \Omega^{\mu_n}) \).

Clearly, \( \Omega^{\sigma_1 \ldots \sigma_k} \mathcal{N}_1^\sigma \ldots \mathcal{N}_n^\sigma \). For each \( \mathcal{N}_1^\sigma \ldots \mathcal{N}_n^\sigma \).

We can define an encoding \( \langle - \rangle : \{ \sigma \} \to \mathbb{N} \) from tokens of the coherence space \( \{ \sigma \} \) to natural numbers as:

- \( \langle \mathcal{N} \rangle = n \) if \( \sigma = \iota \);
- \( \langle (a_1, a_2) \rangle = \langle (a_1), \langle a_2 \rangle \rangle \) if \( \sigma = \tau_1 \to \sigma_2 \).

It provides an enumeration of the tokens of our model. Remark that the Theorem 2 implies that there is a family of terms \( \text{Sgl}^{\lambda \iota} : \sigma \) (short for singleton) being an enumeration of terms that can be interpreted on (single) tokens. Concretely, we define \( \text{Sgl}^{\lambda \iota} : \sigma \) by mutual induction with terms \( \text{Chk}^{\lambda \iota} : \sigma \to \iota \) that checks whether a token is included in the operational behavior of a term typed \( \sigma \).

**Definition 11.** *The terms \text{Sgl}(\sigma) : \sigma \text{ and } \text{Chk}(\sigma) : \sigma \to \iota \text{ are defined by mutual induction on } \sigma.*

If \( \sigma = \iota \), \( \text{Sgl}(\iota) = \emptyset \) and \( \text{Chk}(\iota) = \lambda y. \text{if} (\text{N} \equiv y) \emptyset \Omega^i \).

If \( \sigma = \sigma_1 \to \sigma_2 \), let \( \sigma_2 = \tau_1 \to \ldots \to \tau_k \to \iota \) for some \( k \), without loss of generality, then \( \text{Sgl}(\sigma) \) is

\[
\lambda \xi^{\sigma_1} \ldots \lambda \xi^{\sigma_k} . \text{id} \left( \text{Chk}^{\lambda \iota} (\text{Chk}(\sigma_1) \text{f} \text{Sgl}(\sigma_2) \ldots \text{Sgl}(\sigma_k) \Omega^{\tau_1} \ldots \text{Sgl}(\sigma_k)) \emptyset \Omega^i \right)
\]

and \( \text{Chk}(\sigma) = \lambda x. \text{if} (\text{Chk}(\sigma_1) (\text{f} \text{Sgl}(\sigma_2) \ldots \text{Sgl}(\sigma_k))) \emptyset \Omega^i \).
In $\text{Chk}_n(\sigma)$ we use $\sigma$ as a short for $\sigma \to \iota$. As an instance, if $n = [n_1, n_2]$ then the term $\text{Sg}_n^{1\to\iota}$ is operationally equivalent to the term $\lambda x. \ell f (x = n_1) \ n_2 \ \Omega'$, while the term $\text{Chk}_n^{1\to\iota}$ is operationally equivalent to the term $\lambda f^{1\to\iota} \ell f (f \ n_1 = n_2) \ n \ \Omega'$.

Lemma 4. Let us fix a type $\sigma$. If $a \in \|\sigma\|$ and $n = \{a\}$, then:

1. $[\text{Sg}_n^{1\to\iota}] \rho = \{a\}$;
2. if $[\text{Chk}_n^{1\to\iota}] \ N \rho = \{0\} \rho$ then $a \in \|N\| \rho$.

4.2 Fix-point operational equivalence

In order to simplify the reasoning about programs, we introduce a non-standard notion of operational equivalence.

Definition 12. Let $M^\sigma, N^\sigma \in \text{SIPCF}$, such that $\text{SFV}(M), \text{SFV}(N) \subseteq \{F_1^{n_1}, \ldots, F_m^{n_m}\}$.

- $M \lesssim_\sigma N$ whenever, for all $P_1^{n_1}, \ldots, P_m^{n_m}$, for all $C[\sigma]$ s.t. $C[M[P_1/F_1, \ldots, P_m/F_m]] \in \mathcal{P}$ if $C[N[P_1/F_1, \ldots, P_m/F_m]] \Downarrow n$.
- $M \sim_\sigma N$ if and only if $M \lesssim_\sigma N$ and $N \lesssim_\sigma M$.

It is easy to verify that $\lesssim_\sigma$ is a preorder and $\sim_\sigma$ is an equivalence. Note that the comparison between the fix-point operational equivalence and the standard one is not immediate. Indeed, proving that the two coincide corresponds to prove that $\sim_\sigma$ is also a congruence. For instance, let us assume that both $M^\sigma, N^\sigma$ contain just one free variable $F^{\sigma'}$. If $M \not\sim_\sigma N$ then it is not easy to build contexts $C[\sigma]$ and $\mu F . C[\sigma]$ (depending from the common substitution to $F$) such that $C[M] \Downarrow n$ and $C[\mu F . C[N]] \Downarrow \not\exists$, i.e. $M \not\equiv_\sigma N$.

The standard interpretation is correct with respect to the fix-point operational equivalence too.

Proposition 1. Let $M^\sigma, N^\sigma \in \text{SIPCF}$. $\|M\| = \|N\| \Rightarrow M \sim_\sigma N$.

Proof. Suppose $\text{SFV}(M), \text{SFV}(N) \subseteq \{F_1^{n_1}, \ldots, F_m^{n_m}\}$; let $C[\sigma]$ be a context and let $P_1, \ldots, P_m$ be closed terms such that $C[M[P_1/F_1, \ldots, P_m/F_m]] \in \mathcal{P}$. Suppose $C[N[P_1/F_1, \ldots, P_m/F_m]] \Downarrow n$; then $C[M[P_1/F_1, \ldots, P_m/F_m]] = \|n\|$ by Adequacy; but $[C[N[P_1/F_1, \ldots, P_m/F_m]]] = \|n\|$ by hypothesis, thus $C[N[P_1/F_1, \ldots, P_m/F_m]] \Downarrow n$ by Adequacy. By definition of $\approx_\sigma$ the proof is done.

For clarity, we anticipate that the fix-point equivalence coincide with the standard contextual equivalence (see Sect. 7).

4.3 Full Abstraction Counterexample

We use the second-order gustave-or operator $\text{G\ö}r$: $(\iota \to \iota) \to (\iota \to \iota) \to (\iota \to \iota) \to \iota$, introduced in [30], and equipped with the following evaluation.

\[
\begin{array}{lll}
\frac{}{P_0 \Downarrow 0} & (G_0) & \frac{}{P_0 \Downarrow 1} & (G_1) \\
\frac{P_1 \Downarrow 0}{\text{G\ö}r^=} P_0 \Downarrow 0} & (G_2) & \frac{P_1 \Downarrow 1}{\text{G\ö}r^=} P_0 \Downarrow 1} & (G_3) \\
\frac{P_2 \Downarrow 0}{\text{G\ö}r^=} P_2 \Downarrow 0} & (G_4) & \frac{P_2 \Downarrow 1}{\text{G\ö}r^=} P_2 \Downarrow 1} & (G_5)
\end{array}
\]

The operator $\text{G\ö}r$, non-deterministically provides inputs to its three branches and it looks for their outputs (ex-ante, it is not possible to choose inputs). As soon as the evaluation terminates by using a rule, the other rules cannot terminate anywise, i.e. ex-post the evaluation determines a unique rule which converges. The operator $\text{G\ö}r$ is the operational counterpart of the following clique

\[
g = \left\{ \begin{array}{c}
((0, 0), (1, 0), (0, 1), 0) \\
((0, 1), (0, 0), (1, 0), 1) \\
((1, 0), (0, 1), (0, 0), 2) \\
((1, 1), (1, 1), (1, 1), 3)
\end{array} \right\}
\]

The following result has been proved in [30].

Theorem 4. $\text{G\ö}r$ is not $\text{SIPCF}$-definable
\[
\begin{align*}
\Gamma \cap \Delta &= \emptyset & \Delta \vdash \ell = f_{\sigma_1}^a, \ldots, f_{\sigma_k}^a & \Gamma_1 \vdash N_1 : \sigma_1, \ldots, \Gamma_k \vdash N_k : \sigma_k & \Delta \vdash \ell : \tau & (\text{let-tor}) \\
\end{align*}
\]

Let \( f_{\sigma_1}^a, \ldots, f_{\sigma_k}^a \) be pre-terms such that \( f_{\sigma_1}^a, \ldots, f_{\sigma_k}^a : \Sigma \). It is easy to see that \( [M_1] \rho \neq [M_2] \rho \), by taking \( \rho(f) = g \) as defined in Equation 3. Instead, \( M \sim \rho \) since to separate the two terms a term behaving like \( G \text{bar} \) is necessary. However, such a term is not definable in \( S\text{PCF} \) as shown by Theorem 4.

### 5 The extended language

To recover the problem presented in the previous section, \( S\text{PCF} \) extends the \( S\text{PCF} \) language by means of a new \( \text{let-tor} \) operator.

**Definition 13.** The \( S\text{PCF} \) pre-terms are defined by extending the grammar of pre-terms as follows:

\[
M ::= M_1 \ldots M_n, \text{let } f_1^{a_1} = M_1, \ldots, f_k^{a_k} = M_1 \text{ infor } M_2 \text{ infor } M_3 \text{ infor } M_4
\]

The terms of \( S\text{PCF} \) are the pre-terms typable by using the type system in Table 1.a extended by the rules in Table 2.a.

The \( \text{let-tor} \) generalizes the behavior of the \( G \text{bar} \) operator presented in the previous section by forcing a linear evaluation. It is worth noting that the contexts of terms \( M_1 \) in the \( \text{let-tor} \) rule are managed in an additive way, contracting common (ground, linear and stable) variables. However, note that a \( \text{let-tor} \) binds all the linear variables in its three branches. So, a form of syntactic linearity (by slice [19]) for linear variables is preserved.

**Definition 14.** A slice of a term \( M \) is a new term where:

- each subterm \( \text{if } \) \text{ P } \text{ M } \text{ H } \text{ of } \text{ M } \) is replaced by, either \( (\lambda z \cdot M_0) \) \text{ P } \text{ or } \( (\lambda z \cdot M_1) \) \text{ P } \text{ where } z \text{ is a fresh variable;}  
- each subterm \( \text{let } f_1 = N_1, \ldots, f_k = N_k \text{ infor } M_1 \text{ infor } M_2 \text{ infor } M_3 \) is replaced by, \( (\lambda f_1 \ldots f_k, M_1)N_1 \ldots N_k \text{ where } 1 \leq i \leq 3 \).

Morally, a slice of a term chooses one branch of \( \text{let } \) and one branch of \( \text{let-tor} \). Indeed, a slice contains neither \( \text{let } \) nor \( \text{let-tor} \). It is easy to verify that each linear variable \( f^{a} \) occurs in a slice of a term \( M \) at most once. This says that \( S\text{PCF} \) is syntactically linear by slices. However, as expected, a term containing \( n \) \( \text{let } \) and \( m \) \( \text{let-tor} \) has \( 2^\alpha 3^\beta \) slices which can eventually coincide.

In order to deal with the \( \text{let-tor} \) operator without violating the semantic linearity we need a careful evaluation of terms, this is described using the terms \( S\text{gl}_m^a \) and \( \text{Chk}_m^{(c)} \) introduced in the previous section.
Definition 15. The evaluation relation $\downarrow \subseteq \mathcal{P} \times \mathcal{N}$ for $\mathbb{S}$PCF programs is the smallest relation satisfying the rules in Table 1.b extended by the rules in Table 2.b.

The evaluation rules for $\text{let{-}lor}$ are patterns for infinite rules, likewise the rule for which? (see previous sections) and for $\exists$ in [33]. The evaluation of the $\text{let{-}lor}$ operator is obtained by three distinct rules that explores pairwise the $\text{let{-}lor}$ branches. The evaluation of a $\text{let{-}lor}$ can be performed only in the case two between $N_1, N_2$ and $N_3$ evaluate to the values 0 and $m + 1$ respectively by using a single tuple (i.e., a token) of the traces in the $\text{let{-}lor}$-argument $N_i$, for $i \leq k$. For instance, the (\text{let{lor}gor} $\downarrow \mathcal{P} \times \mathcal{N}$ can be applied in the case $N_1[N_1/f_1, \ldots, N_k/f_k] \downarrow 0$ and $N_2[N_1/f_1, \ldots, N_k/f_k] \downarrow m$ and the same single information of each $N_j$ coded on numeral $n_j$ is used in both evaluations. The check of this constraints is the motivation for introducing the terms $\text{Sgl}_{\sigma}^{\omega}$ and $\text{Chk}_{\sigma}^{\omega}(\varphi)$ above. Albeit ex-ante we don’t know what is the right pair of $\text{let{-}lor}$ branches to evaluate, they are established during the course of the evaluation, so ex-post only one of the three rules can converge. The interpretation of the $\text{let{-}lor}$ operator follows these ideas.

Definition 16. Let $M^\rho, N^\rho \in \mathbb{S}$PCF, and $\rho \in \text{Env}$. The linear interpretation $[M^\rho] : \text{Env} \to Cl([\sigma])$ is defined by the equations in Table 1.c extended by the ones for the $\text{let{-}lor}$ and $\text{exists}$ in Table 2.c.

It is not difficult to see that the correctness w.r.t. the linear model (Theorem 1) still holds for $\mathbb{S}$PCF. For the technical details of this proof, see Appendix A.1.

5.1 $\mathbb{S}$PCF. program examples

We show how to use the $\text{let{-}lor}$ operator in order to program in $\mathbb{S}$PCF, the operator $\text{Glor}$ presented in Section 4. This should so suggest how to recover the counterexample to full abstraction.

In fact, we show something more. We describe how to program a family of terms $\text{Glor}^L$ that generalize the behavior of the operator $\text{Glor}$. The terms $\text{Glor}^L$, are parametrized over an ordered list $L$ of four numerals $k_0, k_1, k_2, k_3$ representing the output returned in the case the assumptions of one of the rule $(G_1), (G_2), (G_3)$ or $(G_4)$ respectively, are satisfied. So, in particular $\text{Glor}^L$-$\text{Glor}$ when $L = \{0, 1, 2, 3\}$ is considered. In order to proceed in a modular way, we introduce also a notation in order to consider restrictions of $\text{Glor}^L$ which use only a subset of the four rules. Precisely, a $\bullet$ in the list $L$ is used to denote the operator obtained omitting the corresponding rules. For instance, $\text{Glor}^{L_0}$ where $L_0 = \{0, 2, 3, \bullet\}$ is defined as

$$
\lambda f_1 f_2 f_3 \text{let } g_1 = f_1, g_2 = f_2, g_3 = f_3 \text{ in } \text{for } \begin{cases}
\text{Glor}^{\ast \ast \ast \ast} & \text{Glor}^{\ast \ast \ast \ast} \text{g}_1 \text{g}_2 \text{g}_3 \text{g}_3 \text{g}_3

\end{cases}
$$

All the $\text{Glor}^L$ defined by just two rules, i.e. for $L$ containing two occurrences of $\bullet$, can be defined likewise. Thus, $\text{Glor}^L_{\omega}$ with parameter $L_1 = \{1, \bullet, 0, 0\}$ can be defined using the $\text{let{-}lor}$ operator as

$$
\lambda f_1 f_2 f_3 \text{let } g_1 = f_1, g_2 = f_2, g_3 = f_3 \text{ in } \text{for } \begin{cases}
\text{Glor}^{\ast \ast \ast \ast} & \text{Glor}^{\ast \ast \ast \ast} \text{g}_1 \text{g}_2 \text{g}_3

\end{cases}
$$

and $\text{Glor}^{L_3}$ with parameter $L_2 = \{\bullet, 0, 3, 4\}$ can be defined as

$$
\lambda f_1 f_2 f_3 \text{let } g_1 = f_1, g_2 = f_2, g_3 = f_3 \text{ in } \text{for } \begin{cases}
\text{Glor}^{\ast \ast \ast \ast} & \text{Glor}^{\ast \ast \ast \ast} \text{g}_1 \text{g}_2 \text{g}_3

\end{cases}
$$

Finally, $\text{Glor}^{L_3}$ with parameter $L_3 = \{0, 1, 2, 3\}$ can be defined as

$$
\lambda x_1 x_2 x_3 \text{let } g_1 = f_1, g_2 = f_2, g_3 = f_3 \text{ in } \text{for } \begin{cases}
\text{Glor}^{\ast \ast \ast \ast} & \text{Glor}^{\ast \ast \ast \ast} \text{g}_1 \text{g}_2 \text{g}_3

\end{cases}
$$

For every parameter $L$, all the $\text{Glor}^L$ can be built analogously. As expected, the $\text{let{-}lor}$ operator is fundamental for the above construction.

We give here also a flavor on how to use the which? operator to define programming constructs useful to collect runtime information. These programming constructs (called $\text{@wh?}^{\varphi}$) are a generalization of the operators introduced in [30, Section 3.1] and will be used in the next Section to prove the Finite Definability Theorem 6. Let us introduce the operators $\text{@wh?}^{\varphi} : (\sigma \to \tau) \to \sigma \to \tau$, with the following operational semantics:

$$
\left[ (M^{\omega \sigma} \text{Sgl}_{\sigma}^{\omega}) P_1 \ldots P_k \downarrow M \right] \frac{\text{Chk}_{\sigma}^{\omega}(N) \downarrow 0}{(\text{@wh?}^{\varphi}^{\omega \sigma} N^{\omega \sigma} \text{N}) P_1 \ldots P_k \downarrow [\text{N}, \text{M}]}
$$

12
Observe that $(@\text{wh}?)_\mathcal{N}P_1 \ldots P_k \downarrow$ if and only if $\mathcal{N}P_1 \ldots P_k \downarrow$. This control operator gives back the result $\mathcal{N}$ of the evaluation of $\mathcal{N}P_1 \ldots P_k$ together with the numeral $\mathcal{N}$ encoding the part of the trace of $\mathcal{N}$ used for the evaluation. This information will be essential in the proof of the Finite Definability Theorem 6.

We now want to show how to program $@\text{wh}?$ in $\mathcal{S}\mathcal{P}\mathcal{C}\mathcal{F}$; the formal proof is slightly involved, so, we give here some examples to understand how this can be done. First, observe that the term $\lambda x.\lambda z.\text{which}(\lambda h^{\sigma_0}.f \ (h \ x))$ implements $@\text{wh}?$.

Informally, the idea of this program is that which? can use the variable $h^{\sigma_0}$ as an observer, to retrieve the value associated to $x$ during a specific evaluation. Suppose now that we want to build a term behaving as $@\text{wh}?$, $\downarrow$. Let $P^i$ be the following term, which is such that $f^{(i-\sigma_0)}_1, g^{\sigma_0}, h^{\sigma_0}_1, m^{\sigma_0}_2 \vdash P^i$:

$$f \left( \lambda x.\pi_1(h_2(\text{@wh}?(\lambda h^{\sigma_0}.f \ (g(z)) \ (x))) \right)$$

In the term above, $h_1$ and $h_2$ play the role of observers. Besides, observe that we have two occurrences of the terms $\pi_1$ as subterm of $P$: this because the two $@\text{wh}?$ give back a number encoding a pair, whose first component is the result of evaluation of their arguments. Finally, observe that the first occurrence of $@\text{wh}?$ (going from left to right) is used in order to retrieve the value of the bound variable $x'$, while the second occurrence of $@\text{wh}?$ is used to retrieve the evaluation of $g$.

Let $M^{(i-\sigma_0)}$ and $N^{\sigma_0}$ such that $M \Downarrow \mathcal{N}$. Consider the term $Q = \text{which}(\lambda h^{\sigma_0}_1.\text{which}(\lambda h^{\sigma_0}_2.M/f[M/g])$. Let us observe that $Q \Downarrow \mathcal{N}$ where $\mathcal{N} = \left[ \left[ \left[ [n, m, m_3], [m_2, m_2] \right] \right] \right]$ where $m_3$ is the argument passed by $M$ to $N$ during the evaluation and $m_2$ is the result of the evaluation of $m_3$. Thus, if we let $L_3 = \lambda w.\left[ \pi_1(w), \pi_2(\pi_2(w)), \pi_2(\pi_2(w)) \right]$, then the term implementing $@\text{wh}?$ is the following

$$L_3(\text{which}(\lambda h^{\sigma_0}_1.\text{which}(\lambda h^{\sigma_0}_2.M))$$

**Theorem 5.** $@\text{wh}?$ is $\mathcal{S}\mathcal{P}\mathcal{C}\mathcal{F}$, programmable.

**Proof.** In the following, we show that, for all $\pi$, there is a term in $\mathcal{S}\mathcal{P}\mathcal{C}\mathcal{F} + \text{which.}$ behaving in the same way as $@\text{wh}?$, $\downarrow$; the proof is by induction on $\pi$.

**Case $\pi = \sigma$.** Let us consider the following term implementing $@\text{wh}?$:

$$\lambda x.\lambda g^{\sigma_1} \ldots g^{\sigma_k}.\text{which}(\lambda h^{\sigma_0}.f \ (h \ x) g_1 \ldots g_k)$$

Observe that $(@\text{wh}? \ M N) P_1 \ldots P_k \Downarrow [n, m]$ if and only if $M \left( (\lambda x.\lambda g^{\sigma} g_1 \ldots g_k) N \right) P_1 \ldots P_k \downarrow n$ if and only if $M \Downarrow = P_1 \ldots P_k \Downarrow$ as required.

**Case $\pi = \pi_1 \rightarrow \sigma$.** Suppose by induction that we are already able to implement both $@\text{wh}?$ and $@\text{wh}?$, $\downarrow$. Suppose $\sigma_2 = \mu_1 \rightarrow \ldots \rightarrow \mu_p \rightarrow \sigma$. Let $L_1^{(i-\sigma_0)}$ be the following term, which is such that $h^{\sigma_0}_1, x^{\sigma_0_\tau} \vdash L_1 : \sigma \rightarrow \tau$.

$$\lambda x. (\lambda y^{\sigma_1} \lambda z^{\mu_1} \ldots z^{\mu_p} \pi_1(h_2(\text{@wh}?^\sigma_1 (g^{\sigma_0}.w) (f y)) z_1 \ldots z_p))$$

Let $L_2$ be the following term, which is such that $g^{\sigma_0}, h^{\sigma_0}_2 \vdash L_2 : \sigma_1$.

$$\lambda z^{\sigma_1} \lambda z^{\mu_1} \ldots z^{\mu_p} \pi_1(h_2(\text{@wh}?^{\sigma_1} (g^{\sigma_0}(z)) z_1 \ldots z_p))$$

Let $L_3^{(i-\sigma_0)}$ be the following term

$$\lambda w.\left[ \pi_1(\pi_1(w)), \pi_2(\pi_2(w)), \pi_2(\pi_2(w)) \right]$$

The term implementing $@\text{wh}?$ is the following:

$$\lambda x.\lambda g^{\sigma_0}.\lambda g^{\sigma_1} \ldots g^{\sigma_k}.L_3(\text{which}(\lambda h^{\sigma_0}.\text{which}(\lambda h^{\sigma_0}.L_2 g_1 \ldots g_k)))$$

Observe that $@\text{wh}?$ $M^{(i-\sigma_0)} N^{\sigma_0} P_1 \ldots P_k \Downarrow [n, m]$ if and only if there is $n_3$ such that

which$(\lambda h^{\sigma_0}_1. L_1[N/x][g_1^{(i-\sigma_0)}]/[h_1] L_2[N/g] P_1 \ldots P_k) \Downarrow n_2$

where $m = [\pi_2(\pi_2(n_3)), \pi_2(n_1)]$ and $n = \pi_1(n_2)$. This is equivalent to the fact that there are $m_1, m_2$ such that $n_2 = [m_2, m_1]$ and

$L_1[N/x][g_1^{(i-\sigma_0)}/[h_1] L_2[N/g][g_1/m_1]/[h_2] P_1 \ldots P_k \Downarrow n_2$

where $m = [\pi_2(m_1), \pi_2(n_1)]$ and $n = m_2$. Suppose that $L'$ is the following term, which is such that $y^{\sigma_1} \vdash L : \iota$.

$$\lambda z^{\mu_1} \ldots z^{\mu_p} \pi_1(\text{@wh}?^{\sigma_1} (N \ y)) z_1 \ldots z_p$$

1For this specific case, the first occurrence of $@\text{wh}?$ is sufficient for the goal. We aim just to suggest how to prove that $@\text{wh}?$ is syntactic sugar.
Thus the statement above is equivalent to the fact that

$$M(\lambda y^n z^1_{i-1} \ldots z^p_{i-1}, \pi_1{((\@\text{wh}? \circ (\lambda \omega^\sigma. w) \,(L)) \,z_1 \ldots z_p)}) \, P_1 \ldots P_k \downarrow m_2$$

Observe that for suitable $R^i_1 \ldots R^p_1$ we know that $(@\text{wh}? \circ (\lambda \omega^\sigma. w) \,(L)) \, R_1 \ldots R_p \downarrow m_1$, so by inductive hypothesis we know that $Sg \, ((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow m_1$ and that $\text{Chk}((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow \emptyset$. So the statement above is equivalent to the fact that

$$M(\lambda y^n z^1_{i-1} \ldots z^p_{i-1}, \pi_1{((\@\text{wh}? \circ (\lambda \omega^\sigma. w) \,(\varepsilon) \, y) \,z_1 \ldots z_p)}) \, P_1 \ldots P_k \downarrow m_2$$

Observe again that the same $R^i_1 \ldots R^p_1$ as above and a suitable $R^*$ are such that $(\circ (\lambda \omega^\sigma. w) \,(\varepsilon)) \, R_1 \ldots R_p \downarrow m_1$. So again by inductive hypothesis we know that $N \, Sg\, ((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow m_1$ and $\text{Chk}((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow \emptyset$. Thus the above statement is equivalent to the fact that

$$M \, P_1 \ldots P_k \downarrow m_2$$

as required. It is not difficult to show that $M \, Sg\, ((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow m_2$ and $\text{Chk}((\circ (\lambda \omega^\sigma. w) \,(L)) \, z_1 \ldots z_p) \downarrow \emptyset$. So the proof is done.

6 Finite Definability and Full Abstraction

In this section we prove that the linear model $L$ is fully abstract with respect to $S/P\text{CF}^\ast$. This result relies on the completeness of the linear interpretation with respect to the operational semantics and on the definability of all the finite cliques by means of $S/P\text{CF}^\ast$ terms. In particular, in a consequence we prove the completeness with respect to the Fix-Point equivalence. In the next section (Proposition 2 and Theorem 8) we then prove that the Fix-Point and the Standard operational equivalences coincide. From this, full abstraction holds also with respect to the latter.

Finite definability asserts that all finite cliques can be defined by means of $S/P\text{CF}^\ast$ terms. Our proof follows a standard scheme for proofs of this kind, e.g. [33],[29]. Non trivial uses of $@\text{wh}?$ and $\&\text{-}\&\text{for}\;\text{ctor}$ constructors are needed in the inductive high-order steps.

Definition 17. Let $u$ be a finite clique of a coherence space in $L$. A term $M$ defines $u$ if and only if $[M] = u$. The class of closed terms having $u$ as interpretation is denoted by $\{u\}$, hence $[u] = \{M \, | \, [M] = u\}$. Moreover, $[u] = M$ is used as an abbreviation for $M \in \{u\}$.

By abuse of notation, in the following we denote $[x]$ a term $M$ such that $M = [x]$. In the following, if $i \in \{1,2\}$ and $k \leq 0$ we denote $\pi^k_i(n)$ to be $\pi_i$ applied $k$ times to $n$. To prove definability, we use the following auxiliary lemma.

Lemma 5. Let $(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ be $\tau_1 \to \ldots \to \tau_1 \to \emptyset$ and $k \leq 0$. Then:

- $(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ is $\emptyset$ if and only if $a_k \leftarrow b_k$.
- $(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, b_4, b_5, b_6)$ is $\emptyset$ if and only if $a_k \leftarrow b_k$.

A measure is needed in the next theorem, the RK of a type is inductively defined as: $\text{RK}(i) = 1; \text{RK}(\sigma \to \tau) = \text{RK}(\sigma) + \text{RK}(\tau)$.

Theorem 6 (Finite Definability). If $u \in C/\text{fin}(\{\sigma\})$ then there exists a closed $M \in S/P\text{CF}^\ast$ such that $M = [u]$.

Proof. Let $\sigma = \tau_1 \to \ldots \to \tau_k \to \emptyset$ for some $k \geq 0$. The proof is by induction on the triple $\langle \text{RK}(\sigma), k, [u] \rangle$ ordered in a lexicographic way. The cases $\text{RK}(\sigma) = 1$ and $\text{RK}(\sigma) = 2$ are easy.

- Consider $\text{RK}(\sigma) = 1$, then $\sigma = \emptyset$ and $[\sigma] = N$. Thus, $\Omega$ and numerals define all possible finite cliques, since $C/\text{fin}(N) = \emptyset \cup \{n / n \in [N]\}$.

- Consider $\text{RK}(\sigma) = 2$, then $\sigma = \emptyset \to \emptyset$.

1. If $[u] = \emptyset$ then $u = \emptyset$ is defined by $\Omega^{\text{-}\text{fin}}$.

2. If $[u] = \emptyset$, then $u = u' \cup \{a, b\}$ for $a, b \in N$. By induction hypothesis we have $[u']$, hence $[u] = \lambda z. \text{if } (z = [a] \, b) \, ([u'] z)$.

- Consider $\text{RK}(\sigma) \geq 3$ and $k = 1$, then $\sigma = \tau \to \emptyset$ with $\tau = \sigma_1 \to \ldots \sigma_r \to \emptyset$.

In general, we use $[a^1, \ldots, a^n]$ as an abbreviation for $\{[a^1], \ldots, [a^n]\}$.
- If \(|u| = 0\) then \(u = \emptyset\) is defined by \(\Omega^{\tau_1 \cdots \tau_k \cdots \tau_0} \cdot u\).

- If \(|u| = 1\), then \(u = \{ (a, b) \}\) for \(a \in \mathbb{N}\) and \(b \in \mathbb{N}\). Suppose \(a = (a_1, \ldots, a_r, c)\) where \(a_i \in [\sigma_i]\) and \(c \in \mathbb{N}\) (1 \(\leq i \leq r\)). By induction hypothesis we have \([a_1], \ldots, [a_r], [c]\) and \([b]\), hence: \(|u| = \lambda f. \ellif (f[a_1] \cdots [a_r] \neq [c]) \ [b] \Omega^i\).

- If \(|u| > 1\), then \(u = \{(a^0, b^0), \ldots, (a^m, b^m)\}\) for \(a^i \in [\tau_i]\) and \(b^i \in \mathbb{N}\) (0 \(\leq i \leq m\)). Suppose \(a^i = (a^i_1, \ldots, a^i_r, c^i)\) where \(a^i_1 \in [\sigma_j]\) and \(c^i \in \mathbb{N}\) (1 \(\leq j \leq r\)). By Lemma 5.1, we have \(a^b \prec a^h\) (0 \(\leq h \neq k \leq m\)), so by Lemma 5.2 \(a^j_1 \prec a^j_2\) (1 \(\leq j \leq r\)). Hence, take \(v_j = \{ a^j_i \ | 1 \leq i \leq m\}\). Moreover, for sake of simplicity, we write just \(a^j_1\) in place of \(\{a^j_1\}\), i.e. the natural number encoding \(a^j_1\).

By induction hypothesis we have \([v_j]\), \([c^i]\) for every 0 \(\leq i \leq m\), 1 \(\leq j \leq r\) and 1 \(\leq k \leq s\). So, we can define:

\[|u| = \lambda \mathbb{F}^r. (\lambda z^i. \ellif (\pi_i'(z) = c^0 \text{ and } \pi_i^{-1}(\pi_2(z)) = a^0_1 \ldots a^0_i \ldots a^0_2 \ b^0 \text{ and } \ldots ) \ (\ellif (\pi_i'(z) = c^1 \text{ and } \pi_i^{-1}(\pi_2(z)) = a^1_1 \ldots a^1_i \ldots a^1_2 \ b^1 \text{ and } \ldots ) \ (\ellif (\pi_i'(z) = c^2 \text{ and } \pi_i^{-1}(\pi_2(z)) = a^2_1 \ldots a^2_i \ldots a^2_2 \ b^2 \text{ and } \ldots ) \ (\ellif (\pi_i'(z) = c^3 \text{ and } \pi_i^{-1}(\pi_2(z)) = a^3_1 \ldots a^3_i \ldots a^3_2 \ b^3 \text{ and } \ldots ) \ (\cdots ) (\ellif (\pi_i'(z) = c^m \text{ and } \pi_i^{-1}(\pi_2(z)) = a^m_1 \ldots a^m_i \ldots a^m_2 \ b^m \text{ and } \ldots ) ) ) ) )

\]

- Consider \(\mathbb{R}K(\sigma) \geq 3\) and \(k > 1\).

- If \(|u| = 0\), then \(u = \emptyset\) is defined by \(\Omega^{\tau_1 \cdots \tau_k \cdots \tau_0} \cdot u\).

- If \(|u| = 1\), then \(u = \{ (a_1, \ldots, a_k, b) \}\) where \(a_i \in [\tau_i]\) (1 \(\leq i \leq k\)) and \(b \in \mathbb{N}\). Thus, \( |u| = \lambda f_1 \ldots f_k. \ellif (f_1 \simeq [a^1_1]) \ (\ellif (f_1 \simeq [a^2_1]) \ldots (\ellif (f_1 \simeq [a^k_1]) \ldots (\ellif (f_k \simeq [b^1]) \ldots (\ellif (f_k \simeq [b^2]) \ldots ) ) ) )

If \(\tau = \nu_1 \cdots \nu_l \cdots \nu_0\) then we have that \(a^1_1 = (c^1_1, \ldots, c^1_j, c^1_l)\) and \(a^2_1 = (c^2_1, \ldots, c^2_j, c^2_l)\). Since \(a^1_1 \prec a^2_1\), we have that for all \(j \in [1, l]\) the sets \(\{c^1_j, c^2_j\}\) are cliques of lower rank. Thus by inductive hypothesis we have terms \(N_i\) defining them. Thus the term defining \(u\) is the following:

\[|u| = \lambda f_1 \ldots f_k. (\lambda x'. \ellif (\pi_1'x \simeq c^1_1 \text{ and } \pi_1^{-1}(\pi_2x) \simeq c^1_1 \ldots (\ellif (f_1 \simeq [a^1_1]) \ldots ) ) ) )

\]

- If \(|u| > 2\), then \(u = \{ d^1, \ldots, d^m \}\) where \(d^i = (a^i_1, \ldots, a^i_k, b^i)\), \(a^i_j \in [\tau_i]\) and \(b^i \in \mathbb{N}\) (1 \(\leq j \leq m\), 1 \(\leq i \leq k\)). We denote by \(d^i[b]\) the token \((a^i_1, \ldots, a^i_k, b)\). By Lemma 5.1 there exists 1 \(\leq h \leq k\) such that \(a^h_1 \prec a^2_1\), so we can build the following finite cliques:

\[
\begin{array}{ll}
\{d^1[0], d^2[b^2 + 1]\} \\
\{d^1[b^1 + 1], d^2[0] \ | 2 < r \leq m\} \\
\{d^2[0], d^3[b^3 + 1] \ | 2 < r \leq m\} \\
\end{array}
\]
Note that $\|w_s\| < \|w\|$ for $s = 1, 2, 3$. So, by induction hypothesis we have $[w_1], [w_2]$ and $[w_3]$. Hence:

$$[u] = \lambda f_1 \ldots f_k. \text{let } g_1 = f_1, \ldots, g_k = f_k \text{ infer } ([w_1] g_1 \ldots g_k) ([w_2] g_1 \ldots g_k) ([w_3] g_1 \ldots g_k)$$

and this concludes the proof.

The definability of finite cliques is the key ingredient to extend the Stable Closed Completeness Theorem 3 to all the terms of $\$PCF_\gamma$ as follows.

**Theorem 7 (Completeness).** Let $M^\sigma, N^\sigma \in \$PCF_\gamma$.

$$M \sim_\sigma N \Rightarrow [M] = [N]$$

**Proof.** Let $\Gamma \vdash M, N : \sigma$ with $\Gamma \vdash S = \{ F_{\gamma_1}^m, \ldots, F_{\gamma_n}^n \}$ and $\Gamma \vdash \ell, \Gamma \vdash \iota = \{ x_1, \ldots, x_n \}$. Assume $[M] \neq [N]$, then there exists $\rho$ such that $[M] \rho 
 [N] \rho$. By the Separability Lemma 3, there exists a closed term $P^{\sigma \rightarrow \iota}$ such that $F([P])([M] \rho) \neq F([P])([N] \rho)$. By the Finite Definability Theorem 6, for all $f$ in $\Gamma \vdash S$ there is a term $P_1 = [\rho(f_i)]$ and for all $x_1$ in $\Gamma \vdash \ell, \Gamma \vdash \iota$ there is a term $N_1 = [\rho(x_1)]$. So, we can build $C = P(\lambda x_1^m, \ldots, x_n^m. [\sigma] N_1 \cdots N_m)$. Without loss of generality, let us assume $F([P])([M] \rho) = \{ k \}$. By adequacy $C[M[P_1/F_1, \ldots, P_n/F_n]] \Downarrow \bar{k}$ but $C[N[P_1/F_1, \ldots, P_n/F_n]] \Downarrow \bar{\bar{k}}$. This concludes the proof.

By soundness and completeness the full abstraction follows.

**Corollary 2 (Full Abstraction).** Let $M^\sigma, N^\sigma \in \$PCF_\gamma$.

$$M \sim_\sigma N \iff [M] = [N]$$

### 7 Coincidence of operational equivalences

In this section we prove the coincidence between the standard operational equivalence (Definition 9) and the fix-point operational equivalence (Definition 12). Therefore, the full abstraction holds also for the standard operational equivalence and a compositional theory of program equivalence can be effectively defined.

The fix-point equivalence coincides with the denotational equivalence by Corollary 2. The proof follows directly by the correctness of the denotational semantics w.r.t the standard operational equivalence.

**Proposition 2.** Let $M^\sigma, N^\sigma \in \$PCF_\gamma$, $M \sim_\sigma N \Rightarrow M \approx_\sigma N$.

The opposite direction is more difficult and it requires a semantic reasoning. First, we prove an auxiliary result claiming that in a coherence space $X \in \mathcal{L}$ (i.e in our type structure) different from $N$, finite cliques are never maximal w.r.t. set-theoretical inclusion (Corollary 3). Then, we use this fact, together with Adequacy (Theorem 1) and Finite Definability (Theorem 6) to prove the result in the ground case (Lemma 9), which implies the general result (Theorem 8).

The fact that finite cliques in a coherence space $X \in \mathcal{L}$ different from $N$ are never maximal w.r.t. set-theoretical inclusion follows from the next lemmas.

**Lemma 6.** Let $x$ be a non-empty finite set of tokens in $[[\sigma]]$ satisfying

$$\exists a \in x. \forall b \in x. a < b$$

Then, $\exists a' \notin x \forall b \in x. a' < b$.

**Proof.** The proof is by induction on the structure of $\sigma$. The case $\sigma = \iota$ is immediate: indeed, since $x$ is finite, it suffices to choose a number in the set $N \setminus x$ to obtain the result. For the inductive case $\sigma = \sigma_1 \rightarrow \sigma_2$, let $a = (a_1, a_2) \in x$ be an element satisfying (4). This implies that $\forall(b_1, b_2) \in x. a_1 < b_1 \land a_2 < b_2$. So it is possible to build the set $x_2 = \{ b_2 | (b_1, b_2) \in x \}$.

By inductive hypothesis there is $a'_2 \notin x_2$ such that for all $b_2 \in x_2$ we have that $a'_2 < b_2$. Now we can set $a' = (a_1, a'_2)$; it is not difficult to check that for all $b \in x$ we have that $a' < b$. This concludes the proof.

**Lemma 7.** Let $x$ be a non-empty finite set of tokens in $[[\sigma]]$ ($\sigma \neq \iota$) satisfying

$$\exists a \in x. \forall b \in x. a < b$$

Then $\exists a' \notin x \forall b \in x. a' < b$.

---

Note that this fact is not true in the general case of stable functions: for example in the coherence space $!N \rightarrow N$, the finite clique $\{ (0, 0) \}$ is maximal.

---
Proof. The proof is by induction on the structure of \( \sigma \). The base case \( \sigma = \iota \) is vacuously true. For the inductive case \( \sigma = \sigma_1 \rightarrow \sigma_2 \), let \( a = (a_1, a_2) \in x \) be an element satisfying (5). Let us consider the sets \( x_1 = \{ b_1 \mid (b_1, b_2) \in x \land b_1 = a_1 \} \) and \( x_2 = \{ b_2 \mid (b_1, b_2) \in x \land b_2 = a_2 \} \). By Lemma 6, there is \( a'_1 \not\in x_1 \) such that for all \( b_1 \in x_1 \) we have \( a'_1 \sim b_1 \). There are two cases:

1. if \( \sigma_2 = \iota \) then we can set \( a' = (a'_1, k) \) where \( k \) is a randomly chosen natural number.

2. if \( \sigma_2 \neq \iota \) then we can apply inductive hypothesis and find \( a'_2 \in x_2 \) such that for all \( b_2 \in x_2 \) we have \( a'_2 \sim b_2 \). So we can set \( a' = (a'_1, a'_2) \).

In both cases it is not difficult to check that for all \( b \in x \) we have \( a' \sim b \). This concludes the proof. \( \square \)

Corollary 3. Let \( x \in Cl_{fin}(\{\sigma\}) \), with \( \sigma \neq \iota \). Then there is \( a \in [\|\sigma\|] \) such that \( a \not\in x \) and \( x \cup \{a\} \in Cl(\{\sigma\}) \).

Given an environment \( \rho \), a term \( M \), a stable variable \( f^\sigma \) and an infinite clique \( x \in Cl(\{\sigma\}) \), with a slight abuse of notation in the sequel we write \([M]_\rho[f := x] = y\) to denote \( \bigcup_{\nu \in x} [M]_\rho[f := y] \). The next auxiliary lemma will be useful in what follows.

Lemma 8. Let \( M^\sigma, N^\sigma \in SPCF_{a} \) and \( \rho \in Env \). If \( M^\sigma[N^\sigma]m \in SPCF_{a} \), then \([M^\sigma[N^\sigma]]_\rho = \bigcup_{\nu \in \nu} [M]_\rho[N]_\rho \).

Now, we show that by using fixpoints it is possible to build contexts that allow us to discriminate as much as we can do by using substitutions.

Lemma 9. If \( \Gamma \vdash M, N : \iota \) with \( \Gamma |_{\iota} = \emptyset \). \( M \approx N \Rightarrow M \approx N \).

Proof. We prove the contrapositive. Let \( F_1, F_2, \ldots, F_n \) be such that \( SVF(M), SVF(N) \subseteq \{F_1, \ldots, F_n\} \). Let \( C[\iota] \) be a context and \( P \) be closed terms such that \( C[M[P/F]] \not\subseteq \|\iota\| \) and \( C[N[P/F]] \not\subseteq \|\iota\| \). We prove by induction on \( n \) that there is a \( C[\iota] \) such that \( C[M[P/F]] \not\subseteq \|\iota\| \) and \( C[N[P/F]] \not\subseteq \|\iota\| \).

Base case. since \( n = 0 \) the two terms have no free occurrence of stable variables, thus we can take \( C[\iota] = C[\iota] \).

Inductive case. The fix-point (in)equality implies that there is a context \( C[\iota] \) and there are closed terms \( P_1, \ldots, P_{n+1} \), such that \( C[M[P_1/F_1, \ldots, P_{n+1}/F_{n+1}]] \not\subseteq \|\iota\| \) but \( C[N[P_1/F_1, \ldots, P_{n+1}/F_{n+1}]] \not\subseteq \|\iota\| \). Thus, by Correctness, there exists a \( \rho \) such that

\[
[M]_\rho[F_{n+1}] := [P_1]_\rho, \ldots, [F_{n+1}]_\rho := [P_{n+1}]_\rho \neq [N]_\rho[F_{n+1}] := [P_1]_\rho, \ldots, [F_{n+1}]_\rho := [P_{n+1}]_\rho.
\]

In the following, we define \( \rho_1 = \rho[F_1] := [P_1]_\rho, \ldots, [F_n]_\rho := [P_n]_\rho \). Let us observe that both \( [M]_{\rho_1}[F_{n+1}] := [P_{n+1}]_\rho \) and \( [N]_{\rho_1}[F_{n+1}] := [P_{n+1}]_\rho \) are finite sets (in particular, they have at most one element and they cannot be both empty). Thus, without loss of generality, we assume that \( [M]_{\rho_1}[F_{n+1}] := [P_{n+1}]_\rho \). So by Lemma 12 there exists \( x \subseteq_{f_{n+1}} [P_{n+1}]_\rho \) such that

\[
[M]_{\rho_1}[F_{n+1}] \neq [N]_{\rho_1}[F_{n+1}] \subseteq \|\iota\| \cap \|\iota\|.
\]

Since stable variables are never of ground type, we can let \( \sigma_{n+1} = \tau_1 \rightarrow \cdots \rightarrow \tau_m \rightarrow \iota \) and \( m \geq 1 \). Let \( x = \{(a_1^1, \ldots, a_{n+1}^1, h^1), \ldots, (a_1^p, \ldots, a_{n+1}^p, h^p)\} \) with \( p \geq 0 \) and \( a_1^1 \in [\|\tau_1\|], \ldots, a_m^1 \in [\|\tau_m\|], h^1 \in \|\iota\| \) for all \( i \in [1, p] \). Furthermore, by Corollary 3 there is a token \( (a_1^1, \ldots, a_m^1, h^m) \not\in x \) such that \( x^* = x \cup \{(a_1^1, \ldots, a_m^1, h^m)\} \) is still a clique. Observe that it is not restrictive to assume \( h^\sigma \not\in \{h_1, \ldots, h_p\} \), since changing of outputs preserves the coherence of tokens. By Finite Definability, there is a term \( F \) defining the clique \( x^* \). Now, let us consider the context

\[
D = f_{n+1} \lambda x^1 \ldots f_n \lambda x^p (\ldots (\ldots (\ldots (x^1 \ldots (x^p \ldots (x^{p-1} \ldots (x^2 \ldots x^1) \ldots) \ldots) \ldots) \ldots)
\]

Observe that \( D[M], D[N] \in SPCF \), because we hypothesized that \( M, N \) have no free occurrences of linear variables. If \( q_1 = (a_1^1, \ldots, a_m^1, q_m = (a_m^1) \), it is not difficult to see that \( D[M], S_{q_1}(\tau_1), \ldots, S_{q_p}(\tau_p) \rho_1 = [M]_{\rho_1}[F_{n+1}] \neq [N]_{\rho_1}[F_{n+1}] \) which is either empty or it is a singleton \( \{k\}' \) different from \( k \).

By Lemma 12, adequacy and definition of fix-point equivalence (since the number of free stable variables has decreased), to conclude the proof. \( \square \)
The above lemma can be used to prove the general case.

**Theorem 8.** Let \( M^\sigma, N^\sigma \in S\text{PCF}_s \), \( M \equiv_\sigma N \Rightarrow M \equiv_\sigma N \).

**Proof.** We prove the contrapositive statement. Let \( f_1^{\sigma_1}, \ldots, f_n^{\sigma_n} \) be such that \( SFV(M), SFV(N) \subseteq \{ f_1, \ldots, f_n \} \). Let \( C \) be a context and \( P \) be closed terms such that \( C[Mf/P] \downarrow \ n \) and \( C[Nf/P] \uparrow \ n \) for some numeral \( n \). In particular, observe that \( C[M] \not\equiv_\sigma C[N] \), by definition of Fix-Point Operational Equivalence. Moreover, by construction \( C[M] \) and \( C[N] \) have no free occurrence of linear variables. Thus, by Lemma 9 there is a context \( D \) such that \( D[C[M]] \downarrow \ n \) and \( D[C[N]] \uparrow \ n \). So, the proof is done. \( \square \)

**Corollary 4.** The equivalence \( \equiv_\sigma \) is a congruence.

### 7.1 Applicative Operational Equivalence

We conclude the section by defining an applicative operational equivalence obtained by considering only special kinds of contexts (Applicative Contexts) to test the equality of terms.

**Definition 18** (Applicative Operational Equivalence). Let \( M^\sigma, N^\sigma \in S\text{PCF}_s \), such that \( SFV(M), SFV(N) \subseteq \{ f_1^{\sigma_1}, \ldots, f_n^{\sigma_n} \} \).

- \( M \equiv_\sigma^A N \) whenever, for all context \( C \) of the form \( \lambda x.([\sigma]) P_1 \ldots P_n \) and for all closed terms \( L_1^{\sigma_1}, \ldots, L_n^{\sigma_n} \), if \( C[Mf/L] \downarrow \ n \) then \( C[Nf/L] \downarrow \ n \).
- \( M \equiv_\sigma^A N \) if \( M \equiv_\sigma N \) and \( N \equiv_\sigma^A M \).

**Theorem 9.** Let \( M^\sigma, N^\sigma \in S\text{PCF}_s \), \( M \equiv_\sigma N \Rightarrow [M] = [N] \).

**Proof.** Just by observing that the context used in the proof of Theorem 7 is an applicative context. \( \square \)

The applicative equivalence still coincides with the previous ones, so it provides a convenient tool for reasoning on programs.

**Corollary 5** (Equivalences Coincidence). Let \( M^\sigma, N^\sigma \in S\text{PCF}_s \).

\[
M \equiv_\sigma N \iff M \equiv_\sigma N \iff M \equiv_\sigma^A N
\]

### 8 A Tracing Evaluation Semantics

The operational semantics of \( S\text{PCF}_s \), presented in the previous sections is effective, but quite inefficient. Indeed, the rules for \( \text{which?} \) and \( \text{let-f\-for} \) non-deterministically face a potentially infinite number of evaluation branches. Consequently, its bovine implementation should try an exhaustive search of the right branch among the infinite ones. In this section, we introduce a tracing evaluation semantics which is able to drastically prune such infinite-branching search tree.

Roughly, denotational linearity provides the certainty that each term is applied to a unique sequence of arguments. That is, only one token of the corresponding trace is used. So, an efficient evaluation can be obtained by storing an arguments list in an environment \( e \), and when the evaluation of \( N \) is done, by replacing the term in the environment by its trace. This way of evaluating programs is a kind of call-by-need evaluation where the trace of the term is stored instead of the value. In order to simplify the evaluation of the new operators, we record such sequence-information along the evaluation tree. The idea is “to recursively trace” the arguments supplied to functions. The so-obtained pruned search-tree is finite-branching and it induces a clever and more efficient evaluation.

A key role in order to trace all the information is played by the environment.

**Definition 19.** An environment \( e \) is a function from a finite set of variables (ground or linear) to either a term (named subject) or a numeral (named trace).

Let \( x^\sigma \) be a variable in the domain of \( e \). If \( \sigma = \iota \) then \( e(x) \) is always the trace. If \( \sigma \) is an arrow then either \( e(x) \) is a term typed \( \sigma \) (the subject) or it is the trace typed \( \iota \).

We note \( e[z:=M] \) the new environment \( e' \) such that \( e'(z) = M \) and \( e'(y) = e(y) \), for each \( y \neq z \). Last, we note \( e|_{\{x\}} \) the restricted environment obtained by deleting the variable \( x \).

Note that, the environments associate terms to type union, i.e. a variable could refer either to a term or to a numeral (encoding the trace). Moreover, in the case of a ground variable we just need to store a numeral because of the call-by-value policy that makes subjects and traces coincide.

The use of an environment to manage substitutions and the presence of a recursion operator raise the issue of clashes between variable-names. For sake of simplicity, we do not introduce closures. We overcame clashes by assuming a whole bunch of fresh
variables and by doing the appropriate renaming at run-time in the reduction rules. Another reason for following that approach is that fresh variables are anyway necessary to trace the evaluation of which? arguments.

As expected, the rule of our machine will be driven by states (i.e. term in environment); they are denoted by \( \langle M \rangle_e \). Given a state, the environment contains as usual the subterms to be substituted to all the free variables and, sometimes, also some additional variables recording traces. Indeed, during the evaluation, sometimes an operational rule “fetches” an head redex and extends the environment with a fresh variable associated to a new subject. When arguments are supplied, the operational rules carefully substitute the subject by the convenient numeral encoding the trace. In this way, if a variable \( x \) contains a trace then, it has recorded the use done during the evaluation of the subject initially associated to the variable \( x \) itself. For sake of clarity, we explicitly remove not used variables from environments.

We note \( \varnothing \) the empty environment, so if \( M \) is a ground closed term then we can obtain its evaluation, by supplying the state \( \langle M \rangle^{\varnothing} \) to the tracing machine.

**Definition 20.** The tracing evaluation \( \triangledown_T \) is the effective relation from states (ground terms in environments) to states (numerals in environments) defined by the rules of Table 3. If \( \langle M \rangle^{\varnothing} \rightarrow \triangledown_T \langle n \rangle_{e_1} \) then we say that \( M \) converges, and we write simply \( M \downarrow T \), otherwise we say that it diverges, and we write \( M \not\Rightarrow_T \).

We emphasize that all the states in rules of table 3 are driven by the head of a ground term. In order to facilitate the comprehension of the tracing machine we have devised its rules in two parts. The rules in the higher Part (a) take into account all possible shapes of the head of terms, except the case of a head variable that is tackled by the rules in the lower Part (b).

The rule in the higher part are quite easy to understand, so we comment only the rules involving non-standard operators. The rule (a) extends the incoming environment \( e_0 \) by a fresh variable \( h \) associated to the term \( M \) that we plan to trace, and evaluates \( h \) applied to the identity \( I = \lambda x. x \) (recall that if \( \text{which}?(M) \) converges, by rules in the Table 1.c, then \( M \) also converges. The result of this evaluation gives back a state \( \langle n \rangle_{e_1} \) where \( e_1 \) reports the observed trace of \( M \). From this, the result is built. Likewise, the three \( \text{let-for} \) rules start by evaluating a branch. If such a branch converges, the reported traces are used in order to start the evaluation of the other branch. In this case we supply as subjects of the involved variables terms having (exactly) the behavior of

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Hvar)</td>
<td>( e(x) = n ) ( \triangledown_T \langle n \rangle_{e} )</td>
</tr>
<tr>
<td>(Happ)</td>
<td>( \text{H fresh, } \langle M \rangle_{e_1} \rightarrow \triangledown_T \langle n \rangle_{e_2} )</td>
</tr>
<tr>
<td>(Happ)</td>
<td>( \text{H fresh, } \langle M \rangle_{e_1} \rightarrow \triangledown_T \langle n \rangle_{e_2} )</td>
</tr>
<tr>
<td>(Happ)</td>
<td>( \text{H fresh, } \langle M \rangle_{e_1} \rightarrow \triangledown_T \langle n \rangle_{e_2} )</td>
</tr>
</tbody>
</table>

Table 3: Tracing Semantics - Part (a) and Part (b).
the reported trace (recall the Lemma 4). Therefore, a rule converges only in case the two branches do the same observations on the list of arguments supplied to a linear variables.

The lower part of Table 3 is a bit more complex. All the rules fetch head-variables, so the discriminating factor that drives their behavior is found in the shape of the subject associated in the environment to the head variable itself. Indeed, it is easy to see that there is one rule for all possible shape of the “head-subject”. We remark that $\texttt{let-\lor}$ is not taken into account, since a term having a $\texttt{let-\lor}$ as head is certainly typed ground and ground terms are evaluated before to be stored in the environment (because the call-by-value policy).

The rule (Hgvar) is easy. (Hvar) considers the case where the subject associated to the head variable is another variable-name, it forward the evaluation by using the subject. (Happ) is a key rule. It applies in the case the subject associated to the head variable is an application $\texttt{MN}$. In this case, it shift $\texttt{N}$ in the term of the state driving the rules. Such approach allows us to collect sufficient information to report the needed traces. The rules (Hs), (Hp) and (HM) are quite simple. Likewise to the rule (w) presented above, the rule (Hv) is interesting: it uses the information collected by the fresh variable $h$ to produce the right outcome and the trace of the which? itself (since it is a subject of the incoming environment). Finally, the rules (HL\textsubscript{\textgreek{a}}), (HL\textsubscript{\textgreek{a}p}), (HL\textsubscript{\textgreek{a}w}) and (HL\textsubscript{\textgreek{a}c}) consider the cases arising from an head variable associated to an abstraction in the environment. There are four rules depending from two types, i.e. the type of the body of the abstraction and the type of the abstracted variable; mnemonically, the rule’s names use as superscript the type of the variable and as subscript the type of the body. The rules (HL\textsubscript{\textgreek{a}}) and (HL\textsubscript{\textgreek{a}c}) need first to evaluate the ground argument in order to comply the call-by-value policy. All the rules compose the sub-traces information in the rule-premises to obtain the traces that they must provide in the rule-conclusion.

We now prove that the trace evaluation machine is correct and complete with respect to the evaluation semantics presented in Table 1 and Table 2. It is easy to check by induction that the rules of Table 3 preserve the following property: if $\texttt{FV}(\texttt{e}) = \{x_1, \ldots, x_n\}$ for some $n \geq 0$ then such variables are in the domain of $e$. Note that $e$ associates to a variable $x$ a term $\texttt{N}$ that can contain variables which are also in the domain of $e$. We show that we can avoid the use of cyclic environments. Let $e_0$ be an environment such that $\texttt{dom}(e_0) = \{x_1, \ldots, x_n\}$, let $\texttt{M}[e]$ be a state, let $\texttt{N}$ be $\texttt{M}[e(x_1)/x_1, \ldots, e(x_n)/x_n]$. We define $T$ to be the function from state to state, defined as follows: $T(\texttt{M}[e]) = (\texttt{M}[e_1]|_{\texttt{FV}(\texttt{M})})$. We plan to transform a state $\texttt{M}[e]$ in a closed ground term by iterating $T$ until the environments becomes (eventually) empty. If such procedure is terminating then the state is said $\text{acyclic}$.

**Lemma 10.** The evaluation of acyclic states only uses acyclic states.

**Proof.** Since we add only fresh variables to the environments and the environments is always finite.

From now on, we consider only acyclic states. Since $\texttt{M}[e]$ is trivially acyclic, by the above lemma, we can define $T_\infty(\texttt{M}[e])$ to be the closed ground term obtained by iterating $T$ until the environment becomes empty.

**Lemma 11.** Let $\sigma = \sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow \gamma$. If $(\texttt{P}_1 \ldots \texttt{P}_k; e_0) \Downarrow \texttt{M}[e_1]$ then $\Downarrow_T \texttt{M}[e_1]$. And $(\texttt{P}_1 \ldots \texttt{P}_k; e_0) \Downarrow \texttt{M}[e_1].$

**Proof.** Easy by induction on the tracing reduction rules.

We can prove the trace evaluation machine to be correct and complete, by using the previous lemmas.

**Theorem 10.** Let $T_\infty(\texttt{M}[e_0]) = \texttt{P}.$

**Correctness.** If $(\texttt{M}[e_0]) \Downarrow_T \texttt{M}[e_1]$ then $\texttt{P} \Downarrow \texttt{M}[e_1].$

**Completeness.** If $\Downarrow_T \texttt{M}[e_0]$ then $\Downarrow_T \texttt{M}[e_1].$

Note that the tracing evaluation machine is a concrete improve with respect to the evaluation machine presented in Table 1 and Table 2, in the sense that it prunes the infinite branching search trees of the evaluation rules for which? and let-\lor. So, it can be regarded as a guideline for a reasonable implementation.

**Example** Let us see the effectiveness of the tracing machine, by showing the evaluation of the term which?(\texttt{let-\lor}.s(f3)). In the following, we denote with $e$ the environment $[f' = \texttt{I}].$

\[
\begin{align*}
(\texttt{3}(e)) & \Downarrow_T (\texttt{3}(e)) & (\texttt{3}(e)[z = \texttt{3}]) & \Downarrow_T (\texttt{3}(e)[z = \texttt{3}]) & (\texttt{Hgvar}) \\
(\texttt{s}(f'(e)) & \Downarrow_T (\texttt{s}(\texttt{3}'[f' = [\texttt{3}, \texttt{3}]))) & (\texttt{s}(\texttt{3}'[f' = [\texttt{3}, \texttt{3}]))) & (s) \\
(\texttt{h} \downarrow \texttt{3}, [h = \texttt{3}[f' = [\texttt{3}, \texttt{3}]) & \Downarrow_T \texttt{3} \Downarrow_T (\texttt{3}, [h = [\texttt{3}, \texttt{3}, \texttt{4}]) & (\texttt{HL\textsubscript{\textgreek{c}}}) \\
\texttt{which?}(\texttt{let-\lor}.s(f3)) & \Downarrow_T (\texttt{4}, [f' = [\texttt{3}, \texttt{3}]]) & (\texttt{w})
\end{align*}
\]
It can be noted here that the rules \((H\lambda_2)\) and \((H\lambda_1)\) are crucial to trace out the use of terms being substituted to the fresh head variables \(h\) and \(t'\). The rule \((H\lambda_3)\) traces the term \(\lambda f. s(f\bar{3})\) applied to \(I\). The rule \((H\lambda_4)\) traces the term \(I\) applied to \(\bar{3}\).

9 Related works

The study of the relations between languages and models is a classic theme in denotational semantics [28, 15]. The full abstraction for PCF has led to the development of very sophisticated semantics techniques (as [2, 23]) and revealed relevant programming principles and operational constructions. In particular, several works have studied how to extend PCF by different operators in order to achieve full abstraction compared to some classical models. For example, this approach has been followed by Plotkin [33] for the continuous Scott model, by Berry and Curien [9] for the sequential algorithms model, by Abramsky and McCusker [1] for a particular game model, by Longley [26] for the strongly stable model and by Paolini [29] for the stable model. Remark that all higher-type operators introduced in the works above [33, 9, 26, 29] cannot be directly interpreted in the linear model. This motivates our search for new operators.

Many linear languages with different goals have been proposed so far in the literature. Recently, in the studies of syntactical linearity, Alves et al. have proposed several syntactical linear languages in order to characterize different classes of computable functions [3, 4, 5]. Such languages are syntactically linear but does not have extra operators that are key ingredients of \(\lambda\text{PCF}\).

Two PCF-like languages embedding linearity notions have been proposed in [10, 11]. These languages are not denotationally linear, in the sense that not all their closed terms are in correspondence with linear functions of a suitable domain. In particular, they cannot be interpreted in the linear model considered here. Despite this fact, the authors of [10, 11] give some interesting results on the relations between several forms of operational reasoning, in the context of the linear decomposition. Our results on the coincidence of three operational equivalences can be viewed as further contributions in those topics.

10 Conclusions and Future Works

The results presented in this paper are part of a wider project (started with the works [17, 30]) aiming to extend the expressive power of linear programming languages. Our aim is to study particular denotational models embedding a notion of linearity, in order to extract programming languages that are fully abstract or fully complete (universal) with respect to the model, and that have new interesting operational features. Our study is also related to higher-type computability [25, 27], since the higher-type of new operators arising from these analysis.

A first interesting direction is to extend the results obtained in this paper in order to prove the universality of \(\lambda\text{PCF}\) with respect to the linear model, i.e. to find the language able to define all the recursive cliques of the model. Another interesting direction is the study of \(\lambda\text{PCF}\), semantics with respect to other model notions. In particular, we would pursue the study started in [31] about a model for \(\lambda\text{PCF}\), based on linear processes. Moreover, we would pursue the study about categorical models for \(\lambda\text{PCF}\), [18].

Last, we guess that an even more efficient evaluation of \(\lambda\text{PCF}\) is possible. In particular, it would be interesting to study a new evaluation machine having an optimized memory management. That is, an evaluation machine that does not need neither fresh variables nor closures and where the heap will be used only for trace out sub-programs.

References

A Technical Proofs

A.1 Proof of Adequacy Theorem

The following adequacy proof is an adaptation of a proof of Plotkin given in [33].

**Lemma 12.** Let $\mathcal{M}, \mathcal{N} \in \mathcal{S}/\mathcal{PCF}$ and $\rho, \rho' \in \text{Env}$.

1. If $\rho(f) \subseteq \rho'(f)$ for each $f \in \text{SFV}(\mathcal{M})$, then $[\mathcal{M}]\rho \subseteq [\mathcal{M}]\rho'$.

2. If $\mathcal{M}[\mathcal{N}/f] \in \mathcal{S}/\mathcal{PCF}$, then $[\mathcal{M}[\mathcal{N}/f]]\rho = \bigcup_{x \subseteq f} [\mathcal{M}]\rho[f := x]$.

3. If $\mathcal{M}[\mathcal{N}/\pi] \in \mathcal{S}/\mathcal{PCF}$ then there exists a unique $a \in [\mathcal{N}]\rho$ such that $[\mathcal{M}[\mathcal{N}/\pi]]\rho = [\mathcal{M}]\rho[\pi := a]$.

4. $[(\lambda x. M)\mathcal{N}] = [\mathcal{M}\mathcal{N}/\pi]$.

5. $[(\lambda f. M)\mathcal{N}] = [\mathcal{M}\mathcal{N}/f^\sigma]$.

6. $[(\text{id} f \text{ op } L R)] = [\mathcal{L}]$ and $[(\text{id} f \text{ op } n + 1 L R)] = [\mathcal{R}]$.

7. $[\mu f^\sigma. M] = [\mathcal{M}\mu f^\sigma. M/f^\sigma]$.

8. If $\sigma = \tau$, $C[\tau] \in \mathcal{CL}_{x^\sigma}$, $C[M]$, $C[N] \in \mathcal{P}$ and $[\mathcal{M}]\rho = [\mathcal{N}]\rho$ then $[C[M]] = [C[N]]$.

**Proof.** (1.) follows by induction on the structure of $\mathcal{M}$. (2.) and (3.) follows by induction on the structure of $\mathcal{M}$. The proofs of (4.), (5.), (6.) and (7.) follow by definition of interpretation and by points (2.) and (3.). The proof of (8.) follows by induction on the structure of $\mathcal{C}[\tau]$. 

\(\square\)
As usual, we have also a finite approximation theorem for the fixpoint semantics.

**Lemma 13.** $\{\mu F. M^\sigma\} \rho = \bigcup_{n \in \mathbb{N}} \{\mu^n F. M^\sigma\} \rho$, for all $\sigma \in \mathbb{I}$.

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

**Theorem 11.** If $\mathcal{M} \Downarrow n$ then $[\mathcal{M}] = [n]$.

**Proof.** By induction on the derivation of $\mathcal{M} \Downarrow n$. Lemma 12 will be crucial to deal with the various inductive steps. □

**Definition 21.** The “computability predicate” is defined by the following cases.

- Case $\text{FV}(M^\sigma) = \emptyset$.
  - Subcase $\sigma = \iota$: $\text{Comp}(M^\sigma)$ if and only if $[\mathcal{M}] = [n]$ implies $\mathcal{M} \Downarrow n$.
  - Subcase $\sigma = \mu - \tau$: $\text{Comp}(M^\sigma)$ if and only if $\text{Comp}(M^\tau M^\iota)$ for each closed $M^\tau$ such that $\text{Comp}(M^\iota)$.

- Case $\text{FV}(M^\sigma) = \{x_1^{\tau_1}, \ldots, x_n^{\tau_n}\}$, for some $n \geq 1$.
  - If $\text{Comp}(M^\sigma)$ if and only if $\text{Comp}(M[N_1/x_1, \ldots, N_n/x_n])$ for each closed $N^\tau_i$ such that $\text{Comp}(N^\tau_i)$.

Lemma 14 states an equivalent formulation of computability predicate.

**Lemma 14.** Let $M^\sigma_1, \ldots, M^\sigma_m \in \text{S} \downarrow \text{PCF}$ and $\text{FV}(M) = \{x_1^{\tau_1}, \ldots, x_n^{\tau_n}\}$ $(n, m \in \mathbb{N})$. $\text{Comp}(M)$ if and only if $[M[N_1/x_1, \ldots, N_n/x_n]] = [m]$ implies $M[N_1/x_1, \ldots, N_n/x_n] P_1 \ldots P_m \Downarrow n$ for each closed $N^\tau_i$ and $P_j$ such that $\text{Comp}(N_i)$ and $\text{Comp}(P_j)$ where $i \leq n, j \leq m$.

Adequacy follows immediately by next lemmas.

**Lemma 15.**

1. $\text{Comp}([\sigma_1^{(\mathcal{N})}])$

2. Let $N^\sigma$ be a closed term such that $\text{Comp}(N)$. If $[\text{Chk}_N^{(\sigma)}] = [0]$ then $\text{Chk}_N^{(\sigma)} \Downarrow 0$.

**Proof.** By mutual induction on $\sigma$. The base case $\sigma = \iota$ is easy. Let us develop the case $\sigma = \sigma_1 - \sigma_2$. (1) Let $\sigma_2 = \tau_1 - \sigma_2$ and suppose $[\text{Sgl}^{(\sigma_1)}] M N_1 \ldots N_k = [m]$ for all closed terms $M^{\iota_1}, N_1^{\omega_1}, \ldots, N_k^{\omega_k}$ such that $\text{Comp}(M), \text{Comp}(N_1), \ldots, \text{Comp}(N_k)$. By definition of interpretation, this means that

i. $[\text{Chk}_N^{(\sigma_1)}] M \rho = [0] \rho$.

ii. $[\text{Sgl}_N^{(\sigma_1)}] N_1 \ldots N_k \rho = [m] \rho$.

By applying mutual induction on (i) we get $\text{Chk}_N^{(\sigma_1)} \Downarrow 0$, and by applying induction on (ii) we get $\text{Sgl}_N^{(\sigma_1)} \Downarrow 0$. Thus we conclude by applying the evaluation rule (ii). (2) Suppose $[\text{Chk}_N^{(\sigma)}] = [0]$ for a closed term $N$ such that $\text{Comp}(N)$. By definition of interpretation, this means that $[(\text{Chk}_N^{(\sigma)})(\text{NSgl}_N^{(\sigma)}))] \rho = [0] \rho$.

By mutual induction, we have that $\text{Comp}([\text{Sgl}_N^{(\sigma)}]_N^{(\sigma)})$, thus by hypothesis and by definition of computability predicate, we have $\text{Comp}(N_1 \text{Sgl}_N^{(\sigma)}(\sigma_1), \ldots, N_k \text{Sgl}_N^{(\sigma)}(\sigma_k))$. Thus we can apply inductive hypothesis, showing that

(Chk$_N^{(\sigma_1)}(N \text{Sgl}_N^{(\sigma_1)}(\sigma_1)) \Downarrow 0$)

Thus we conclude by applying the evaluation rule (iif). □

**Lemma 16.** If $M^\sigma \in \text{S} \downarrow \text{PCF}$ then $\text{Comp}(M^\sigma)$.

**Proof.** By induction on the shape of $\mathcal{M}$. We detail just the two most interesting cases.

- $M = x$. Let $\sigma = \tau_1 - \sigma_2$ where $m \in \mathbb{N}$. Let $P^\sigma$ and $N_i^\tau$ for $1 \leq i \leq m$ be closed terms such that $\text{Comp}(P^\sigma)$ and $\text{Comp}(N_i^\tau)$. By definition, $\text{Comp}(P^\sigma)$ imply that, if $[P^\sigma] = [n]$ then $P^\sigma \Downarrow n$ by definition of the computability predicate.
• \( M = \text{NP} \). Assume \( N^{\sigma} \) and \( P^\tau \) for types \( \sigma \) and \( \tau \). By induction hypothesis \( \text{Comp}(N^{\sigma}) \) and \( \text{Comp}(P^\tau) \) and the proof follows by definition of the computability predicate.

• \( M = \lambda x. \Omega \). Assume \( x^\mu \) and \( Q^\nu \) for types \( \mu \) and \( \nu \). Let \( FV(M) = \{ x_1^{\mu_1}, \ldots, x_k^{\mu_k} \} \) for \( k \geq 0 \) and \( \tau = \tau_1 \rightarrow \cdots \rightarrow \tau_h \rightarrow \iota \), where \( h \geq 0 \). Let \( \nu_1^{\mu_1}, \ldots, \nu_k^{\mu_k}, \mu_0^\nu, \nu_1^\tau, \ldots, \nu_k^\nu \) be closed terms such that \( \text{Comp}(\nu_i) \) and \( \text{Comp}(\mu_j) \) for \( 1 \leq i \leq k \) and \( 0 \leq j \leq h \) respectively. Thus \( \text{Comp}(Q^\nu [P_0/x][M_1/x_1, \ldots, M_k/x_k]P_1 \ldots P_h) \), since \( \text{Comp}(Q^\nu) \) holds by induction hypothesis. Consider the case \( \mu \neq \iota \) and suppose

\[
\left[ (\lambda x^\mu. Q^\nu) [M_1/x_1, \ldots, M_k/x_k]P_0 \ldots P_h \right] = [n]
\]

Therefore \( [Q^\nu [P_0/x][M_1/x_1, \ldots, M_k/x_k]P_1 \ldots P_h] = [n] \) by Lemma 12 points (5.) and (8.). Thus \( Q^\nu [P_0/x][M_1/x_1, \ldots, M_k/x_k]P_1 \ldots P_h \downarrow \beta \) by induction hypothesis. So, \( (\lambda x^\mu. Q^\nu) [M_1/x_1, \ldots, M_k/x_k]P_0 \ldots P_h \downarrow \beta \) by the evaluation rule \( (\lambda \iota) \). Now, suppose \( \mu = \iota \) and

\[
\left[ (\lambda x^\mu. Q^\nu) [M_1/x_1, \ldots, M_k/x_k]P_0 \ldots P_h \right] = [n],
\]

Since eval is strict, we must have \( [P_0] = [m] \) for some \( m \). But \( \text{Comp}(P_0) \) by induction and \( [P_0] = [m] \) imply \( P_0 \downarrow m \). Hence

\[
[Q^\nu [m/x][M_1/x_1, \ldots, M_k/x_k]P_1 \ldots P_h] = [n]
\]

by induction hypothesis. The proof follows by applying the evaluation rule \( (\lambda) \). For the other case, we conclude again by using inductive hypothesis and the evaluation rule \( (\iota) \).

• \( M = \text{if} \ M_1 M_2 M_3 \). Assume \( FV(M) = \{ x_1^{\mu_1}, \ldots, x_n^{\mu_n} \} \) and let \( P_1^\nu, \ldots, P_n^\nu \) be closed terms such that \( \text{Comp}(P_i) \) for all \( i \). Suppose that

\[
[M[P_1/x_1, \ldots, P_n/x_n]] = [m]
\]

Then, by interpretation, either

\[
[M_1[P_1/x_1, \ldots, P_n/x_n]] = [0]
\]

and

\[
[M_2[P_1/x_1, \ldots, P_n/x_n]] = [m]
\]

or

\[
[M_3[P_1/x_1, \ldots, P_n/x_n]] = [m + 1]
\]

and

\[
[M_4[P_1/x_1, \ldots, P_n/x_n]] = [m]
\]

In the first case we have \( M_1[P_1/x_1, \ldots, P_n/x_n] \downarrow 0 \) and \( M_2[P_1/x_1, \ldots, P_n/x_n] \downarrow m \) by inductive hypothesis; thus we conclude by the evaluation rule \( (\text{if}) \). For the other case, we conclude again by using inductive hypothesis and the evaluation rule \( (\text{if}) \).

• \( M = \mu F. x \). Assume \( N^\sigma \) for some type \( \sigma \). Let \( FV(M) = \{ x_1^{\mu_1}, \ldots, x_k^{\mu_k} \} \) for \( k \geq 0 \) and \( \sigma = \sigma_1 \rightarrow \cdots \rightarrow \sigma_h \rightarrow \iota \), where \( h \geq 0 \). By induction on \( h \), likewise to the corresponding proof of [33]. The case \( h = 0 \) is trivial, so assume \( h \geq 1 \). Assume \( \nu_1^{\mu_1}, \ldots, \nu_k^{\mu_k} \) and \( P_1^\nu, \ldots, P_h^\nu \) be closed terms such that \( \text{Comp}(\nu_i) \) and \( \text{Comp}(P_j) \) for \( 1 \leq i \leq k \) and \( 1 \leq j \leq h \) respectively. Let \( [\mu F. N^\sigma [M_1/x_1, \ldots, M_k/x_k]P_1 \ldots P_h] = [n] \). By Lemma 13, we have

\[
\left[ (\mu F. N^\sigma [N_1/x_1, \ldots, N_k/x_k])P_1 \ldots P_h \right] =
\left[ (\mu F. N^\sigma [M_1/x_1, \ldots, M_k/x_k])P_1 \ldots P_h \right]
\]

for some \( k \in \mathbb{N} \). Thus, \( \mu F. N^\sigma [Q/x_1, \ldots, Q/x_n]P_1 \ldots P_h \downarrow \beta \) by the previous points of this lemma. The proof follows by Lemma 13.?
The linear interpretation is adequate for $\lambda$PCF.

**Proof.** It follows directly by the previous lemma.

\[\square\]