EPiC Series in Computing

Volume 100, 2024, Pages 181–197

Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning

Hybrid Intersection Types for PCF

Pablo Barenbaum[1‡], Delia Kesner[2], and Mariana Milicich[2§]

[1] Universidad Nacional de Quilmes (CONICET), and Instituto de Ciencias de la Computación, UBA
Argentina
[2] Université Paris Cité, CNRS, IRIF
France

Abstract

Intersection type systems have been *independently* applied to different evaluation strategies, such as call-by-name (CBN) and call-by-value (CBV). These type systems have been then generalized to different subsuming paradigms being able, in particular, to *encode* CBN and CBV in a *unique* unifying framework. However, there are no intersection type systems that explicitly enable CBN and CBV to cohabit together, without making use of an encoding into a common target framework.

This work proposes an intersection type system for a specific notion of evaluation for PCF, called PCFH. Evaluation in PCFH actually has a *hybrid* nature, in the sense that CBN and CBV operational behaviors cohabit together. Indeed, PCFH combines a CBV-like behavior for function application with a CBN-like behavior for recursion. This hybrid nature is reflected in the type system, which turns out to be *sound* and *complete* with respect to PCFH: not only typability implies normalization, but also the converse holds. Moreover, the type system is *quantitative*, in the sense that the size of typing derivations provides upper bounds for the length of the reduction sequences to normal form. This first type system is then refined to a *tight* one, offering *exact* information regarding the length of normalization sequences. This is the first time that a sound and complete quantitative type system has been designed for a hybrid computational model.

1 Introduction

Evaluation strategies govern how computation proceeds in programming languages. Two prominent strategies are *call-by-name* (CBN) and *call-by-value* (CBV) [24]. In CBN, arguments are not evaluated before being consumed by functions, while in CBV, arguments must be fully evaluated to *values* before function application can proceed.

These two ways to perform evaluation have their own advantages and drawbacks. CBV evaluation can sometimes be less costly than CBN, since CBV evaluates arguments only once, while CBN may have to evaluate many copies of a single argument. Conversely, sometimes

‡ *Partially funded by project grants PUNQ 418/22 and PICT-2023-602.*

§ *This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 945332.*

CBN evaluation can be less costly than CBV, as CBN does not evaluate an *unused* argument while CBV always evaluates arguments, even if their value is not needed. For instance, if the function g is constantly 0 and Ω is a looping program, then $g(\Omega)$ produces 0 as a result in CBN while CBV evaluation does not terminate.

The main motivation of this work is to enhance our understanding of the *quantitative semantics* of programming language constructs such as inductive data types and recursion. The quantitative semantics of minimalistic languages such as the λ-calculus are relatively well-understood in both the CBN and CBV settings (see [9, 13]), but inductive data types and recursion cannot be expressed directly, and must instead rely on some sort of *encoding*. In this work, we study PCFH, an extension of the λ-calculus that incorporates natural numbers, conditional expressions, and recursion through a fixed-point operator without resorting to an encoding.

Non-idempotent intersection types. Intersection types (IT), pioneered by Coppo and Dezani [10], extend simple types with a new *intersection* type constructor ($\cap$), in such a way that a term is assigned the type $\tau \cap \sigma$ if and only if it is assigned both types τ and σ. Originally, intersection is commutative, associative and, in particular, *idempotent*, *i.e.* τ and $\tau \cap \tau$ are equivalent. An intersection $\tau_1 \cap \ldots \cap \tau_n$ can thus be specified by a *set* $\{\tau_1, \ldots, \tau_n\}$. IT were motivated by semantical concerns. A key property is that typability is preserved by *conversion* (reduction *and* expansion), in contrast with simple types, in which typability is preserved by reduction but not expansion. This technical property allows to study normalization from a type theoretical point of view. In intersection types, a term is typable if and only if it is normalizing, while in simple types, typable terms are normalizing but the converse does not hold.

A more recent line of research is that of *non-idempotent* IT [18, 9], in which $\tau \cap \tau$ is not equivalent to τ. Here, an intersection $\tau_1 \cap \ldots \cap \tau_n$ may be specified by a *multiset* $[\tau_1, \ldots, \tau_n]$. These systems draw inspiration from Linear Logic (LL) [19], as in fact, non-idempotent intersection corresponds to LL's multiplicative conjunction ($\otimes$) [9]. Just like in the idempotent case, non-idempotent IT characterize normalization by means of typability [9]. Moreover, non-idempotent types have some advantages over idempotent ones; one key benefit is their ability to capture *quantitative* information about the dynamic behavior of programs. For example, in non-idempotent IT one can not only prove that a term t is normalizing if and only if it is typable, but also obtain an *upper bound* for the length of the evaluation sequence of t to normal form. Since these systems are inspired by linear logic and resource consumption [18], they are called *quantitative* type systems. Several works have explored non-idempotent intersection type systems in the context of CBN [11, 8] and CBV [13, 3, 21]. In these systems, the quantitative information can usually be recorded using *counters* that decorate typing judgments. For example, a typical judgment may be of the form $\Gamma \vdash^n t : \tau$, meaning that t normalizes to a normal form of type τ, and that the number of reduction steps required to reduce t to normal form is *bounded* by the value of the counter $n \in \mathbb{N}$.

Quantitative properties provided by non-idempotent intersection type systems can be refined using a syntactic property, *tightness*, achieved by using only *minimal* type derivations [2]. Tightness allows to extract from typing derivations not only *upper bounds* but rather *exact* quantitative information. Indeed, in tight derivations, a concluding judgment is still of the form $\Gamma \vdash^n t : \tau$, but now the counter $n \in \mathbb{N}$ indicates that *exactly* n reduction steps are required to reduce t to normal form.

PCF and PCFH. *Programming Computable Functions* (PCF) [25] is a functional programming language designed to serve as a foundational computational model. It extends the simply-

typed λ-calculus with natural numbers, conditional expressions, and general recursion (through a fixed-point operator). PCF is simultaneously minimalistic, expressive, and rigorous, making it an ideal vehicle to study programming language semantics. Historically, it has been used to study the correspondence between operational and denotational semantics, *i.e.*, the *full abstraction* problem [25, 23, 1, 17].

In this work, we study a variant of PCF called PCFH, in which function application and evaluation of conditional expressions follow a CBV discipline. Besides that, the rule to *unfold* a fixed-point is the standard one, namely:

$$\mathsf{fix}\, x.\, t \to t\{x := \mathsf{fix}\, x.\, t\}$$

A noteworthy remark is that the semantics of recursion is akin to CBN rather than to CBV since the fixed-point can make copies of itself, which is *not* a value yet. Given this "mixed" operational behavior, we say that evaluation in PCFH is *hybrid*. The fact that evaluation is hybrid is challenging from the point of view of quantitative semantics [21, 6] because it requires synthesizing characteristics of CBN and CBV in a single system.

A note on nomenclature. Languages closely related to PCFH are sometimes said to be "call-by-value" [12]. As explained above, evaluation in PCFH is actually hybrid because unfolding a fixed-point may produce copies of itself, substituting a variable by an expression that is not a value yet. We write PCFH to highlight the fact that evaluation is hybrid, as indicated by the subscript "H".

Related work. There is a vast amount of literature on PCF. In particular, various notions of evaluation have been proposed for PCF; for instance, [20] provides insights on differences and similarities between CBV semantics and CBN semantics. A probabilistic version of PCF is presented in [17], which follows a CBN discipline but is still hybrid in that it exhibits a CBV behavior for its conditional operator.

Furthermore, some works explore semantical aspects of PCF that are connected to our own. In [14], Ehrhard studies an interpretation of a Call-by-Push-Value (CBPV) λ-calculus with disjoint unions and fixed-points (encompassing PCF) in LL, and it derives an intersection type system from the Scott model of LL. The unfolding of a fixed-point operator can duplicate itself by copying a value that represents its suspended computation. Technically, this is achieved by relying on the of-course (!) modality of LL. The resulting type system is in the style of Coppo–Dezani, with *idempotent* intersections, and thus not quantitative.

Recent work in [15] derives an intersection type system for a differential extension of PCF from a relational semantics. It is shown that typability in this system implies normalization by means of a reducibility argument, but the quantitative aspect is not studied. In contrast, we prove soundness and completeness by elementary means, and we characterize quantitative upper bounds and exact bounds via our non-idempotent type system.

There are other languages in the literature that allow the encoding of CBN/CBV behavior, such as Levy's *call-by-push-value* (CBPV) [22] and Ehrhard and Guerrieri's *Bang Calculus* [16]. They attempt to unify CBN and CBV in a single framework, but they are not hybrid as they only duplicate values. Recent works [21, 6] propose quantitative type systems for *adequate* variants of the Bang Calculus. These works study upper bounds and exact bounds. However, they are not able to express recursion and integers directly, but only through encodings.

Contributions. As a main contribution, we propose a quantitative model for the hybrid calculus PCFH, based on a non-idempotent intersection type system. The type system is proved

to be sound and complete with respect to the operational semantics: it characterizes normalizing terms and provides upper bounds for the length of reductions to normal form. A peculiar feature of this system is that it distinguishes between those variables bound by λ-abstractions from those bound by fixed-point operators. The former may be substituted by *values* while the latter may be substituted by arbitrary *expressions*. Type assignment rules work differently for these two kinds of variables because a value is assigned one type each time it will be used (as in CBV), while an arbitrary expression may be copied to produce many values (as in CBN).

As a second contribution, we refine the type system to obtain a *tight* version, which provides *exact* bounds for reduction lengths to normal form, instead of upper bounds.

To the best of our knowledge, this is the first quantitative type interpretation that is adequate for a hybrid system.

Structure of the paper. Section 2 covers the syntax and operational semantics of PCFH. Section 3 introduces a quantitative type system for PCFH. Finally, the paper concludes by summarizing our work and outlining potential directions for future work.

Proofs. Some proofs in the paper have been ommited, but they can be found in the extended version [5].

2 The PCFH calculus

This section introduces the syntax of PCFH, together with its associated operational semantics, which follows a hybrid evaluation. We characterize the set of normal forms induced by the operational semantics through a grammar (Proposition 2). Moreover, as the evaluation strategy is not necessarily deterministic, we show that it enjoys the Diamond Property (Proposition 3), which implies in particular that all evaluation sequences to normal form have the same length, thus justifying the use of the proposed strategy to study quantitative behaviors.

Given a denumerable set of **variables** $(x, y, z, \ldots)$, we define **terms** $(t, s, u, \ldots)$, **values** $(\mathbf{v}, \mathbf{w}, \ldots)$, and **natural values** $(\mathbf{k}, \mathbf{l}, \ldots)$ using the following grammar:

$$t ::= x \mid \lambda x.\, t \mid t\, t \mid \mathbf{0} \mid \mathbf{S}(t) \mid \mathsf{if}(t, t, x.\, t) \mid \mathsf{fix}\, x.\, t$$

$$\mathbf{v} ::= \lambda x.\, t \mid \mathbf{k} \qquad \mathbf{k} ::= \mathbf{0} \mid \mathbf{S}(\mathbf{k})$$

Terms include **variables** x, **abstractions** $\lambda x.\, t$, **applications** $t\, s$, a **fixed-point operator** $\mathsf{fix}\, x.\, t$, as well as syntax for natural numbers: **zero** $\mathbf{0}$, **the successor constructor** $\mathbf{S}(t)$, and a **conditional operator** $\mathsf{if}(t, s, x.\, u)$. Free and bound occurrences of variables are defined as expected, considering that the free occurrences of x in u are bound in $\mathsf{if}(t, s, x.\, u)$ and in $\mathsf{fix}\, x.\, u$. Terms are considered up to α-renaming of bound variables. We write $t\{x := s\}$ for the capture-avoiding substitution of the free occurrences of x by s in t.

The operational semantics of PCFH is given by a reduction relation $\to_\rho$ indexed by a **rule name** $\rho \in \{\mathtt{B}, \mathtt{I0}, \mathtt{IS}, \mathtt{F}\}$, and it is defined by the following set of rules:

$$\frac{}{(\lambda x.\, t)\, \mathbf{v} \to_{\mathtt{B}} t\{x := \mathbf{v}\}}\ \mathtt{r\text{-}beta} \qquad \frac{}{\mathsf{if}(\mathbf{0}, t, x.\, s) \to_{\mathtt{I0}} t}\ \mathtt{r\text{-}if0}$$

$$\frac{}{\mathsf{if}(\mathbf{S}(\mathbf{k}), t, x.\, s) \to_{\mathtt{IS}} s\{x := \mathbf{k}\}}\ \mathtt{r\text{-}ifS} \qquad \frac{}{\mathsf{fix}\, x.\, t \to_{\mathtt{F}} t\{x := \mathsf{fix}\, x.\, t\}}\ \mathtt{r\text{-}fix}$$

$$\frac{t \to_\rho t'}{t\, s \to_\rho t'\, s}\ \mathtt{r\text{-}appL} \qquad \frac{s \to_\rho s'}{t\, s \to_\rho t\, s'}\ \mathtt{r\text{-}appR}$$

$$\frac{t \to_\rho t'}{\mathbf{S}(t) \to_\rho \mathbf{S}(t')}\,\texttt{r-succ} \qquad \frac{t \to_\rho t'}{\mathsf{if}(t,s,x.\,u) \to_\rho \mathsf{if}(t',s,x.\,u)}\,\texttt{r-if}$$

The resulting evaluation strategy is *closed* (no reduction of terms with free variables) and *weak* (no evaluation under abstractions).

Several variants of PCF can be found in the literature. In this presentation, function application follows a CBV discipline because the argument of an application must be evaluated until it becomes a value, so that the rule `r-beta` can be applied. The conditional operator follows a CBV discipline as well, as the guard must be fully evaluated to a value so that one of the rules `r-if0` or `r-ifS` can be applied. The conditional $\mathsf{if}(t,s,x.\,u)$ returns s if t is zero, and u if t is non-zero, binding x to its predecessor. A consequence of this is that the predecessor function can be defined using the conditional operator; *e.g.* $\mathsf{if}(\mathbf{S}(\mathbf{k}),\mathbf{0},x.\,x) \to_{\texttt{IS}} \mathbf{k}$, a variant which can be found in [17].

To illustrate this strategy, let us evaluate the term $\mathsf{if}(\mathbf{S}(\mathbf{0}),\texttt{id},x.\,\lambda y.\,x)\,\mathbf{S}(\mathbf{S}(\mathbf{0}))$ in one step, where $\texttt{id} := \lambda z.\,z$:

$$\dfrac{\dfrac{}{\mathsf{if}(\mathbf{S}(\mathbf{0}),\texttt{id},x.\,\lambda y.\,x) \to_{\texttt{IS}} (\lambda y.\,x)\{x := \mathbf{0}\}}\,\texttt{r-ifS}}{\mathsf{if}(\mathbf{S}(\mathbf{0}),\texttt{id},x.\,\lambda y.\,x)\,\mathbf{S}(\mathbf{S}(\mathbf{0})) \to_{\texttt{IS}} (\lambda y.\,\mathbf{0})\,\mathbf{S}(\mathbf{S}(\mathbf{0}))}\,\texttt{r-appL}$$

After one more reduction step, by rule `r-beta`, we obtain $(\lambda y.\,\mathbf{0})\,\mathbf{S}(\mathbf{S}(\mathbf{0})) \to_{\texttt{B}} \mathbf{0}$.

Let us write $t \to s$ if $t \to_\rho s$ for some rule name ρ. A term t is **$\to$-reducible** if there exists s such that $t \to s$; and t is **$\to$-irreducible** if t is not $\to$-reducible. For the example above, $\mathbf{0}$ is $\to$-irreducible, while $\mathsf{if}(\mathbf{S}(\mathbf{0}),\texttt{id},x.\,\lambda y.\,x)\,\mathbf{S}(\mathbf{S}(\mathbf{0}))$ is $\to$-reducible.

Terms that cannot be further $\to$-reduced are also called **normal forms**. In the following, we characterize the normal forms of the proposed strategy by means of an inductive definition. Normal forms are of different kinds or *natures*. For instance, $\lambda x.\,x\,x$ is a normal form of *abstraction* nature, while $\mathbf{S}(\mathbf{0})$ is a normal form of *natural number* nature. The preceding normal forms are intuitively "meaningful", and they are said to be *proper*; these normal forms are those that can be successfully typed in a quantitative type system, as we discuss in the next section. In contrast, there are intuitively "meaningless" normal forms such as $\mathbf{S}(\lambda x.\,t)$, or $\mathbf{0}\,t$, or $\mathsf{if}(\lambda x.\,t,s,y.\,u)$. These are said to be *stuck*. Stuck normal forms do not have any meaning, so they cannot be typed in a quantitative type system. We return to this point in the next section.

Formally, we first distinguish between different *natures* which will be used to index the sets of normal forms. Thus, we establish the sets of **proper natures**, $\nu \in \{\mathsf{abs},\mathsf{nat}\}$ and **fallible natures**, $\varepsilon \in \{\mathsf{abs},\mathsf{nat},\mathsf{stuck}\}$.

We write NF_ε to denote the set of **normal forms** indexed by a fallible nature ε, which is defined by the following rules:

$$\frac{}{\lambda x.\,t \in \mathsf{NF}_{\mathsf{abs}}}\,\texttt{nf-abs} \qquad \frac{t \in \mathsf{NF}_\varepsilon \quad s \in \mathsf{NF}_{\varepsilon'} \quad \varepsilon \neq \mathsf{abs}}{t\,s \in \mathsf{NF}_{\mathsf{stuck}}}\,\texttt{nf-app} \qquad \frac{}{\mathbf{0} \in \mathsf{NF}_{\mathsf{nat}}}\,\texttt{nf-zero}$$

$$\frac{t \in \mathsf{NF}_{\mathsf{nat}}}{\mathbf{S}(t) \in \mathsf{NF}_{\mathsf{nat}}}\,\texttt{nf-succ-nat} \qquad \frac{t \in \mathsf{NF}_\varepsilon \quad \varepsilon \neq \mathsf{nat}}{\mathbf{S}(t) \in \mathsf{NF}_{\mathsf{stuck}}}\,\texttt{nf-succ-stuck} \qquad \frac{t \in \mathsf{NF}_\varepsilon \quad \varepsilon \neq \mathsf{nat}}{\mathsf{if}(t,s,x.\,u) \in \mathsf{NF}_{\mathsf{stuck}}}\,\texttt{nf-if}$$

For instance, $\lambda y.\,\texttt{id}\,\texttt{id} \in \mathsf{NF}_{\mathsf{abs}}$ because of rule `nf-abs`, while $(\lambda y.\,y\,\mathbf{0})\,\texttt{id}$ is not in normal form since its left subterm is in normal form but with the proper nature abs.

The results below show that this inductively defined notion of normal form characterizes exactly the set of irreducible terms. This syntactic characterization is a key ingredient to show

that the quantitative type system studied in Section 3 is sound and complete. We start with a simple observation regarding the expected form of normal forms based on their assigned proper natures:

Lemma 1. *Let $t \in \mathsf{NF}_\varepsilon$. Then:*

1. *$\varepsilon = \mathsf{abs}$ if and only if t is of the form $\lambda x.\, s$,*
2. *$\varepsilon = \mathsf{nat}$ if and only if t is of the form $\mathbf{k}$.*

The characterization of normal forms then follows using the previous lemma:

Proposition 2 (Characterization of normal forms)**.** *For any closed term t the following are equivalent:*

1. *There exists a fallible nature ε such that $t \in \mathsf{NF}_\varepsilon$.*
2. *t is $\to$-irreducible.*

Evaluation of applications is non-deterministic, *e.g.* consider $((\lambda x.\, \mathbf{S}(x))\, \mathbf{0})\, (\mathsf{if}(\mathbf{0}, \mathtt{id}, y.\, y\, \mathtt{id}))$. This term reduces in one step to $\mathbf{S}(\mathbf{0})\, (\mathsf{if}(\mathbf{0}, \mathtt{id}, y.\, y\, \mathtt{id}))$ by rules `r-appL` and `r-beta`, and it also reduces in one step to $((\lambda x.\, \mathbf{S}(x))\, \mathbf{0})\, \mathtt{id}$ by rule `r-if0`. Thus, we need to ensure that evaluating terms leads to unique normal forms, despite some form of non-determinism during the computation. To achieve this, we prove the Diamond Property for the relation $\to$, saying that if a term reduces in one step to two different terms t_1 and t_2, then both terms converge in one step to the same common reduct. This is a strong form of confluence, which in particular ensures that all reductions to normal form have the same length:

Proposition 3 (Diamond property)**.** *If $t \to_{\rho_1} t_1$ and $t \to_{\rho_2} t_2$ where $t_1 \neq t_2$ then there exists t' such that $t_1 \to_{\rho_2} t'$ and $t_2 \to_{\rho_1} t'$.*

Let us take the term of the example above to illustrate this property:

$$\begin{array}{ccc} ((\lambda x.\, \mathbf{S}(x))\, \mathbf{0})\, (\mathsf{if}(\mathbf{0}, \mathtt{id}, y.\, y\, \mathtt{id})) & \xrightarrow[\mathtt{B}]{} & \mathbf{S}(\mathbf{0})\, (\mathsf{if}(\mathbf{0}, \mathtt{id}, y.\, y\, \mathtt{id})) \\ \Big\downarrow{\scriptstyle \mathtt{I0}} & & \Big\downarrow{\scriptstyle \mathtt{I0}} \\ ((\lambda x.\, \mathbf{S}(x))\, \mathbf{0})\, \mathtt{id} & \xrightarrow[\mathtt{B}]{} & \mathbf{S}(\mathbf{0})\, \mathtt{id} \end{array}$$

Moreover $\mathbf{S}(\mathbf{0})\, \mathtt{id} \in \mathsf{NF}_{\mathsf{stuck}}$ by rule `nf-app`.

3 A Quantitative Type System for PCFH

In this section, we introduce system $\mathcal{H}$ (for *hybrid*), a non-idempotent intersection type system for PCFH. A type derivation for a term in system $\mathcal{H}$ provides upper bounds for its normalization sequences. Moreover, by considering only *tight* typing derivations, we obtain *exact* bounds. System $\mathcal{H}$ aligns with other formulations of non-idempotent intersection type systems and satisfies fundamental properties such as Subject Reduction (Lemma 11) and Subject Expansion (Lemma 14). Remarkably, this system captures the combined essence of the two underlying evaluation strategies found in PCFH, which are CBV and CBN.

The remainder of the section unfolds as follows: Section 3.1 defines the set of types and typing rules governing $\mathcal{H}$. Section 3.2 delves into the proof of soundness and completeness of

$\mathcal{H}$ with respect to the evaluation strategy PCFH. For this, we show two quantitative results: Theorem 17 provides upper bounds for normalization sequences for any kind of type derivations, while Theorem 18 provides exact bounds for normalization sequences for those type derivations that are tight.

3.1 The Type System $\mathcal{H}$

Recall that values in PCFH are either abstractions or natural values of the form $\mathbf{S}(\mathbf{S}(\dots\mathbf{S}(\mathbf{0})))$. A value in PCFH may be *used* zero, one, or many times. For example, the underlined identity function in $(\lambda f.\, f\,(f\,\mathbf{0}))\,\underline{\mathtt{id}}$ is used twice in a reduction to normal form since it is bound to the variable f that appears twice, and the underlined zero in $(\lambda x.\, \mathsf{if}(x, \mathbf{0}, y.\, y))\,\underline{\mathbf{0}}$ is used exactly once in a reduction to normal form because it is bound to a single occurrence of x tested for zero equality.

A noteworthy characteristic of non-idempotent intersection type systems is their ability to capture the several roles a single expression may play in different contexts, so in these type systems *terms do not necessarily have a unique type*. Indeed, in system $\mathcal{H}$, values are given a *multitype* of the form $\mathcal{T} = [\tau_1, \dots, \tau_n]^\nu$, which consists of a multiset of types $[\tau_1, \dots, \tau_n]$ decorated with a nature ν, and corresponding to the (non-idempotent) intersection $\tau_1 \cap \dots \cap \tau_n$. Each τ_i corresponds to one *use* of the value. System $\mathcal{H}$ captures the hybrid behavior of PCFH by splitting the typing information into two parts, called the *typing context* and the *family context*. The typing context contains typing information for variables bound by *abstractions* and *conditional operators*. These variables behave like in CBV, in the sense that they can only be substituted by *values*, and thus they are assigned a multitype. The family context contains typing information for variables bound by *fixed-point operators*. These variables behave like in CBN, as they can be substituted by *unevaluated expressions*. Since an unevaluated expression may be copied many times and each copy produces a value, these variables are assigned a multiset of multitypes $\mathcal{F} = \{\!\{\mathcal{T}_1, \dots, \mathcal{T}_n\}\!\}$, called a *multitype family*.

We note $[\dots]$ for the multisets of types and $\{\!\{\dots\}\!\}$ for the multisets of multitypes. This is just for visual clarity and to emphasize the different roles they play in the type system. However, both notations denote multisets, and both behave like non-idempotent intersections.

Formally, types of $\mathcal{H}$ are given by the following grammar:

(**Types**)	τ	::=	$\mathbb{A} \mid \mathbb{N}$
(abs-**Types**)	$\mathbb{A}$	::=	$\mathcal{T}^? \to \mathcal{T}$
(nat-**Types**)	$\mathbb{N}, \mathbb{M}, \dots$	::=	$\mathbb{0} \mid \mathbb{S}(\mathcal{N})$
(**Optional Multitypes**)	$\mathcal{T}^?$	::=	$\bot \mid \mathcal{T}$
(**Multitypes**)	$\mathcal{T}, \mathcal{S}, \mathcal{U}, \mathcal{R}, \dots$	::=	$\mathcal{A} \mid \mathcal{N}$
(abs-**Multitypes**)	$\mathcal{A}, \mathcal{B}$	::=	$[\mathbb{A}_i]^{\mathsf{abs}}_{i\in I}$
(nat-**Multitypes**)	$\mathcal{N}, \mathcal{M}, \mathcal{P}, \dots$	::=	$[\mathbb{N}_i]^{\mathsf{nat}}_{i\in I}$
(**Multitype Families**)	$\mathcal{F}$	::=	$\{\!\{\mathcal{T}_i\}\!\}_{i\in I}$

A ν-multitype is a multiset of ν-types decorated with a nature ν. *E.g.*, $[\mathbb{0}, \mathbb{0}, \mathbb{S}([\,]^{\mathsf{nat}})]$ is a multiset of nat-types, and $[\mathbb{0}, \mathbb{0}, \mathbb{S}([\,]^{\mathsf{nat}})]^{\mathsf{nat}}$ is a nat-multitype. The decoration is important to distinguish $[\,]^{\mathsf{nat}}$ from $[\,]^{\mathsf{abs}}$, so that a clear distinction is made between variables that are bound to natural numbers from those that are bound to abstractions.

Note that a multitype is either a nat-multitype or an abs-multitype. A ν-multitype is said to be of nature ν. Two multitypes are **compatible** if they are of the same nature. For example, $[\mathbb{0}]^{\mathsf{nat}}$ is compatible with $[\mathbb{S}([\,]^{\mathsf{nat}})]^{\mathsf{nat}}$ but not with $[\mathcal{T}^? \to \mathcal{T}]^{\mathsf{abs}}$. This notion is extended to optional multitypes by declaring that $\mathcal{T}^?$ and $\mathcal{S}^?$ are compatible if either of

them is $\bot$ or if they are compatible multitypes. The **sum** of compatible multitypes is defined by $[\mathbb{A}_1,\dots,\mathbb{A}_n]^{\mathsf{abs}} + [\mathbb{A}_{n+1},\dots,\mathbb{A}_{n+m}]^{\mathsf{abs}} = [\mathbb{A}_1,\dots,\mathbb{A}_{n+m}]^{\mathsf{abs}}$ and, similarly, $[\mathbb{N}_1,\dots,\mathbb{N}_n]^{\mathsf{nat}} + [\mathbb{N}_{n+1},\dots,\mathbb{N}_{n+m}]^{\mathsf{nat}} = [\mathbb{N}_1,\dots,\mathbb{N}_{n+m}]^{\mathsf{nat}}$. This operation is extended to compatible optional multitypes in the following way: $\bot + \bot = \bot$ and $\bot + \mathcal{T} = \mathcal{T}$ and $\mathcal{T} + \bot = \mathcal{T}$. Note that $\bot$ is the neutral element of the sum.

A **family context**, written $\Phi, \Psi, \dots$, is a function mapping variables to multitype families such that $\Phi(x) \neq \{\!\{\,\}\!\}$ for finitely many variables x. The **sum** of family contexts is defined by $(\Phi+\Psi)(x) = \Phi(x) \oplus \Psi(x)$, where $\oplus$ is the sum of multisets of multitypes, defined as the disjoint union of multisets. The **domain** of a family context Φ is defined as $\mathsf{dom}(\Phi) \stackrel{\text{def}}{=} \{x \mid \Phi(x) \neq \{\!\{\,\}\!\}\}$, and $\varnothing$ denotes the empty family context, mapping every variable to $\{\!\{\,\}\!\}$.

A **typing context**, written $\Gamma, \Delta, \dots$, is a function mapping variables to optional multitypes such that $\Gamma(x) \neq \bot$ for finitely many variables x. Two typing contexts Γ, Δ are compatible if $\Gamma(x)$ and $\Delta(x)$ are compatible for every variable x. The **sum** of compatible typing contexts is defined by $(\Gamma + \Delta)(x) = \Gamma(x) + \Delta(x)$. The **domain** of a typing context Γ is defined as $\mathsf{dom}(\Gamma) \stackrel{\text{def}}{=} \{x \mid \Gamma(x) \neq \bot\}$, and $\varnothing$ denotes the empty typing context, mapping every variable to $[\,]$.

A binary relation of **subsumption** $\mathcal{T}_1^? \lhd \mathcal{T}_2$ is defined by two cases, stating that $\bot \lhd [\,]^\nu$ and $\mathcal{T} \lhd \mathcal{T}$ hold. Subsumption is used to introduce a controlled form of weakening in the system (see Example 3.1 below).

A **multi-counter** $\mathtt{m}$ is a multiset of rule names, whose cardinality is denoted by $\#(\mathtt{m})$. **Typing judgments** are of the form $\Phi; \Gamma \vdash^{\mathtt{m}} t : \mathcal{T}$, where Φ is a family context, Γ is a typing context, t is a term and $\mathcal{T}$ is a multitype. Moreover, each judgment typing a term t is decorated with a multi-counter $\mathtt{m}$, which traces all the rewriting rules that are used to evaluate the term t to normal form. This means in particular that $\#(\mathtt{m})$ reflects the length of evaluation sequences to normal form. The decision to use counters that are multisets of rule names instead of natural numbers is to store more precise information. Moreover, we use a multiset rather than a 4-uple of distinct counters, as used for example in [21], to avoid unnecessarily inflating the notation.

As mentioned before, variables can receive two distinct assignments depending on whether they occur in the family context or the typing context. When a variable is assigned a multitype family in the family context, it will be involved in the evaluation of a fixed-point operator, meaning that it is intended to be substituted by an unevaluated expression, like in CBN. Instead, when a variable is assigned a multitype in the typing context, it is intended to be substituted by a value, like in CBV. There is an invariant in $\mathcal{H}$ stating that family contexts and typing contexts do not share variables, *i.e.* $\mathsf{dom}(\Phi) \cap \mathsf{dom}(\Gamma) = \varnothing$. When studying properties for $\mathcal{H}$ we assume implicitly that this invariant holds.

Typing rules for the typing system are the following:

$$\frac{}{\varnothing; x : \mathcal{T} \vdash^{[]} x : \mathcal{T}}\ \mathtt{t\text{-}var_1} \qquad \frac{}{x : \{\!\{\mathcal{T}\}\!\}; \varnothing \vdash^{[]} x : \mathcal{T}}\ \mathtt{t\text{-}var_2}$$

$$\frac{(\Phi_i; \Gamma_i, x : \mathcal{T}_i^? \vdash^{\mathtt{m}_i} t : \mathcal{S}_i)_{i\in I}}{+_{i\in I}\Phi_i; +_{i\in I}\Gamma_i \vdash^{+_{i\in I}\mathtt{m}_i} \lambda x.\, t : [\mathcal{T}_i^? \to \mathcal{S}_i]^{\mathsf{abs}}_{i\in I}}\ \mathtt{t\text{-}abs}$$

$$\frac{\Phi_1; \Gamma_1 \vdash^{\mathtt{m}_1} t : [\mathcal{T}^? \to \mathcal{S}]^{\mathsf{abs}} \qquad \mathcal{T}^? \lhd \mathcal{T} \qquad \Phi_2; \Gamma_2 \vdash^{\mathtt{m}_2} s : \mathcal{T}}{\Phi_1 + \Phi_2; \Gamma_1 + \Gamma_2 \vdash^{[\mathtt{B}]+\mathtt{m}_1+\mathtt{m}_2} t\, s : \mathcal{S}}\ \mathtt{t\text{-}app}$$

$$\frac{}{\varnothing; \varnothing \vdash^{[]} \mathbf{0} : [\mathbb{0}]^{\mathsf{nat}}_{i\in I}}\ \mathtt{t\text{-}zero} \qquad \frac{\Phi; \Gamma \vdash^{\mathtt{m}} t : \mathcal{N} \qquad \mathcal{N} = +_{i\in I}\mathcal{N}_i}{\Phi; \Gamma \vdash^{\mathtt{m}} \mathbf{S}(t) : [\mathbb{S}(\mathcal{N}_i)]^{\mathsf{nat}}_{i\in I}}\ \mathtt{t\text{-}succ}$$

$$\frac{\Phi_1;\Gamma_1 \vdash^{\mathtt{m}_1} t : [\mathbb{0}]^{\mathsf{nat}} \qquad \Phi_2;\Gamma_2 \vdash^{\mathtt{m}_2} s : \mathcal{T}}{\Phi_1+\Phi_2;\Gamma_1+\Gamma_2 \vdash^{[\mathtt{I0}]+\mathtt{m}_1+\mathtt{m}_2} \mathsf{if}(t,s,x.\,u) : \mathcal{T}}\ \mathtt{t\text{-}ifZero}$$

$$\frac{\Phi_1;\Gamma_1 \vdash^{\mathtt{m}_1} t : [\mathbb{S}(\mathcal{N})]^{\mathsf{nat}} \qquad \mathcal{N}^? \lhd \mathcal{N} \qquad \Phi_2;\Gamma_2, x : \mathcal{N}^? \vdash^{\mathtt{m}_2} u : \mathcal{T}}{\Phi_1+\Phi_2;\Gamma_1+\Gamma_2 \vdash^{[\mathtt{IS}]+\mathtt{m}_1+\mathtt{m}_2} \mathsf{if}(t,s,x.\,u) : \mathcal{T}}\ \mathtt{t\text{-}ifSucc}$$

$$\frac{\Phi, x : \{\!\{\mathcal{T}_i\}\!\}_{i\in I};\Gamma \vdash^{\mathtt{m}} t : \mathcal{S} \qquad (\Phi_i;\Gamma_i \vdash^{\mathtt{m}_i} \mathsf{fix}\,x.\,t : \mathcal{T}_i)_{i\in I}}{\Phi +_{i\in I} \Phi_i;\Gamma +_{i\in I} \Gamma_i \vdash^{[\mathtt{F}]+\mathtt{m}+_{i\in I}\mathtt{m}_i} \mathsf{fix}\,x.\,t : \mathcal{S}}\ \mathtt{t\text{-}fix}$$

The set I in the typing rules above can be empty. For example, when $I = \varnothing$ in rule `t-abs`, we obtain a judgment of the form $\varnothing;\varnothing \vdash^{[]} \lambda x.\,t : []^{\mathsf{abs}}$ with no premises. As we discussed at the beginning of this subsection, this result is because the abstraction is not used in the program. When $I = \varnothing$ in rule `t-fix`, it means that there are zero uses of the recursive calls in the program.

Moreover, $I = \varnothing$ in `t-succ` we may encounter a special case that is the following:

$$\frac{\Phi;\Gamma \vdash^{\mathtt{m}} t : []^{\mathsf{nat}}}{\Phi;\Gamma \vdash^{\mathtt{m}} \mathbf{S}(t) : []^{\mathsf{nat}}}\ \mathtt{t\text{-}succ} \qquad \frac{\Phi;\Gamma \vdash^{\mathtt{m}} t : []^{\mathsf{nat}}}{\Phi;\Gamma \vdash^{\mathtt{m}} \mathbf{S}(t) : [\mathbb{S}([]^{\mathsf{nat}})]^{\mathsf{nat}}}\ \mathtt{t\text{-}succ}$$

Note that there may be several ways to split a nat-multitype $\mathcal{N}$ into a sum $+_{i\in I}\mathcal{N}_i$ of nat-multitypes $\mathcal{N}_i$ in rule `t-succ`. Therefore, this rule is non-deterministic when read from top to bottom.

A **(typing) derivation** is a tree obtained by applying the rules above. We write $\pi \rhd \Phi;\Gamma \vdash^{\mathtt{m}} t : \mathcal{T}$ when π is a derivation of the judgment $\Phi;\Gamma \vdash^{\mathtt{m}} t : \mathcal{T}$.

As anticipated in the introduction, the hybrid CBN/CBV operational nature of PCFH is reflected in the quantitative type system. This is apparent by comparing the typing rules of system $\mathcal{H}$ with the rules of quantitative type systems for CBN/CBV in the literature. To see this formal resemblance more clearly, consider for example the CBN quantitative type system $\mathcal{N}$ and the CBV quantitative type system $\mathcal{V}$, both described in [21]. Rule $\mathtt{t\text{-}var}_1$ of $\mathcal{H}$ types variables involved in CBV computations, aligning with rule $\mathtt{var}_\mathtt{c}^{\mathcal{V}}$ of $\mathcal{V}$. Conversely, rule $\mathtt{t\text{-}var}_2$ types variables involved in CBN computations, corresponding to rule $\mathtt{var}_\mathtt{c}$ of $\mathcal{N}$. The typing rule for abstractions, `t-abs`, coincides with rule $\mathtt{abs}_\mathtt{c}^{\mathcal{V}}$ of $\mathcal{V}$. The multitype in the conclusion of rule `t-abs` corresponds to the number of times an abstraction is *used*, as in quantitative type systems for CBV [13], rather than to the number of times the abstraction duplicates its argument, as in quantitative type systems for CBN [18, 9], reflecting the fact that function applications in PCFH are evaluated like in CBV. Conversely, in rule `t-fix` the cardinality of the multitype family $\{\!\{\mathcal{T}_i\}\!\}_{i\in I}$ corresponds intuitively to the number of recursive calls, *i.e.* to the number of times that the fixed-point operator duplicates itself, which is similar to the typing rule of an application in CBN.

The goal of providing PCFH with such system is to characterize normalization of its evaluation strategy, so, as we briefly mentioned above, the purpose of multi-counters $\mathtt{m}$ in $\mathcal{H}$ is to provide *upper bounds* for the number of evaluation steps to normal form by means of its cardinality. However, we would also like system $\mathcal{H}$ to go beyond this, by giving *exact bounds*, and for that, we make use of special derivations called *tight*. A multitype is **tight** if it is of the form $[]^{\nu}$. A derivation $\pi \rhd \Phi;\Gamma \vdash^{\mathtt{m}} t : \mathcal{T}$ is **tight** if Φ and Γ are both empty and if the multitype $\mathcal{T}$ is tight.

For example, the following derivation types the term $\lambda x.\, x\,\mathbf{0}$:

$$\dfrac{\dfrac{\dfrac{}{\varnothing; x : [[\,]^{\mathsf{nat}} \to [\,]^{\mathsf{abs}}]^{\mathsf{abs}} \vdash^{[\,]} x : [[\,]^{\mathsf{nat}} \to [\,]^{\mathsf{abs}}]^{\mathsf{abs}}}\,\texttt{t-var}_1 \qquad \dfrac{}{\varnothing; \varnothing \vdash^{[\,]} \mathbf{0} : [\,]^{\mathsf{nat}}}\,\texttt{t-zero}}{\varnothing; x : [[\,]^{\mathsf{abs}} \to [\,]^{\mathsf{nat}}]^{\mathsf{abs}} \vdash^{[\mathtt{B}]} x\,\mathbf{0} : [[\,]^{\mathsf{abs}}]^{\mathsf{abs}}}\,\texttt{t-app}}{\varnothing; \varnothing \vdash^{[\mathtt{B}]} \lambda x.\, x\,\mathbf{0} : [[[\,]^{\mathsf{nat}} \to [\,]^{\mathsf{abs}}]^{\mathsf{abs}} \to [\,]^{\mathsf{abs}}]^{\mathsf{abs}}}\,\texttt{t-abs}$$

and the derivation is not tight since the resulting multitype for the term is not empty. Moreover, the multi-counter is not an exact bound but rather an upper bound, given that this term is a normal form.

On the other hand, we can also build a *tight* derivation for the same term, ending in $\varnothing; \varnothing \vdash^{[\,]} \lambda x.\, x\,\mathbf{0} : [\,]^{\mathsf{abs}}$ by rule `t-abs` with no premises. Furthermore, the multi-counter here provides an exact bound.

Controlled weakening As mentioned in the introduction, due to their quantitative nature, non-idempotent intersection type systems are *resource-aware*. In consonance with this, system $\mathcal{H}$ is *linear*, hence every single type assumption is used only once. However, in two specific points a controlled form of *weakening* is needed, namely in the `t-app` and `t-ifSucc` rules, reflected in the premises that contain the subsumption relation $\mathcal{T}^{?} \lhd \mathcal{T}$. This allows a variable to be bound to a value which is then *discarded* without being used. Observe that the subsumption relation $\lhd$ is used for controlling the CBV-like behavior. For example, in a term like $(\lambda x.\, \mathbf{0})\, t$, the argument t must be fully evaluated to a value before the application can proceed. Since system $\mathcal{H}$ intends to characterize normalization, this means that t must be typed to ensure that it is normalizing. If the type of t is $\mathcal{T}$ then, in principle, x should also be of type $\mathcal{T}$. But by linearity, $\mathbf{0}$ is only typable under the empty typing contexts. Assuming that t is a closed term of type $[\,]^{\mathsf{nat}}$, a type derivation can be given as follows:

$$\dfrac{\dfrac{\dfrac{}{\varnothing; \varnothing, x : \bot \vdash^{[\,]} \mathbf{0} : [\,]^{\mathsf{nat}}}\,\texttt{t-zero}}{\varnothing; \varnothing \vdash^{[\,]} \lambda x.\, \mathbf{0} : [\bot \to [\,]^{\mathsf{nat}}]^{\mathsf{abs}}}\,\texttt{t-abs} \qquad \bot \lhd [\,]^{\mathsf{nat}} \qquad \begin{array}{c}\vdots\\ \varnothing; \varnothing \vdash^{\mathtt{m}} t : [\,]^{\mathsf{nat}}\end{array}}{\varnothing; \varnothing \vdash^{[\mathtt{B}]+\mathtt{m}} (\lambda x.\, \mathbf{0})\, t : [\,]^{\mathsf{nat}}}\,\texttt{t-app}$$

Note that $\bot \lhd [\,]^{\mathsf{nat}}$ implies the hypothesis $x : \bot$ appears on the typing judgment of $\mathbf{0}$, while it forces the argument t on the right premise to be typable. Also, recall that $\varnothing, x : \bot$ is equal to $\varnothing$.

Let us now give an example of a type derivation for fixed-point operators, so let $t := (\mathsf{fix}\, f.\, \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))))\, \mathbf{S}(\mathbf{0})$, where the left subterm is a function returning the double of any natural number: *e.g.* t computes the double of $\mathbf{S}(\mathbf{S}(\mathbf{0}))$. Let $\mathcal{N} := [\mathbb{S}([\mathbb{0}]^{\mathsf{nat}})]^{\mathsf{nat}}$, $\mathcal{M} := [\mathbb{0}]^{\mathsf{nat}}$, $\mathcal{A} := [\mathcal{N} \to [\,]^{\mathsf{nat}}]^{\mathsf{abs}}$ and $\mathcal{B} := [\mathcal{M} \to [\,]^{\mathsf{nat}}]^{\mathsf{abs}}$.

A tight type derivation for t follows:

$$\dfrac{\pi_{\mathrm{rec1}} \qquad \dfrac{\dfrac{}{\varnothing; \varnothing \vdash^{[\,]} \mathbf{0} : \mathcal{M}}\,\texttt{t-zero}}{\varnothing; \varnothing \vdash^{[\,]} \mathbf{S}(\mathbf{0}) : \mathcal{N}}\,\texttt{t-succ}}{\varnothing; \varnothing \vdash^{[\mathtt{B},\mathtt{F},\mathtt{IS},\mathtt{B},\mathtt{F},\mathtt{IO}]} (\mathsf{fix}\, f.\, \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))))\, \mathbf{S}(\mathbf{0}) : [\,]^{\mathsf{nat}}}\,\texttt{t-app}$$

Note that the argument $\mathbf{S}(\mathbf{0})$ is typed with the singleton nat-multitype $\mathcal{N}$. This reflects the single use of $\mathbf{S}(\mathbf{0})$, as it corresponds to a single occurrence of n tested for zero equality. Moreover,

its subterm $\mathbf{0}$ is assigned the singleton nat-multitype $\mathcal{M}$, since it is bound to a single occurrence of n which is also tested for zero equality in the base case of the recursive function.

The typing derivation for the left subterm (the fixed-point operator) follows:

$$\pi_{\mathrm{rec1}} := \left(\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing; n : \mathcal{N} \vdash^{[]} n : \mathcal{N}}\,\texttt{t-var}_1 \qquad \pi_{\mathrm{cond}}}{f : \{\!\{\mathcal{B}\}\!\}; n : \mathcal{N} \vdash^{[\mathtt{IS},\mathtt{B}]} \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : [\,]^{\mathsf{nat}}}\,\texttt{t-ifSucc}}{f : \{\!\{\mathcal{B}\}\!\}; \varnothing \vdash^{[\mathtt{IS},\mathtt{B}]} \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : \mathcal{A}}\,\texttt{t-abs} \qquad \pi_{\mathrm{rec2}}}{\varnothing; \varnothing \vdash^{[\mathtt{F},\mathtt{IS},\mathtt{B},\mathtt{F},\mathtt{IO}]} \mathsf{fix}\, f.\, \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : \mathcal{A}}\,\texttt{t-fix} \right)$$

on which the function $\lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m)))$ must be typed considering that n shall be bound to $\mathbf{S}(\mathbf{0})$, so that n is of type $\mathcal{N}$. Moreover, f is bound to the result of the recursive call, typed in π_{rec2}:

$$\pi_{\mathrm{rec2}} := \left(\dfrac{\dfrac{\dfrac{\dfrac{}{\varnothing; n : \mathcal{M} \vdash^{[]} n : \mathcal{M}}\,\texttt{t-var}_1 \qquad \dfrac{}{\varnothing; \varnothing \vdash^{[]} \mathbf{0} : [\,]^{\mathsf{nat}}}\,\texttt{t-zero}}{\varnothing; n : \mathcal{M} \vdash^{[\mathtt{IO}]} \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : [\,]^{\mathsf{nat}}}\,\texttt{t-ifZero}}{\varnothing; \varnothing \vdash^{[\mathtt{IO}]} \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : \mathcal{B}}\,\texttt{t-abs}}{\varnothing; \varnothing \vdash^{[\mathtt{F},\mathtt{IO}]} \mathsf{fix}\, f.\, \lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m))) : \mathcal{B}}\,\texttt{t-fix} \right)$$

This derivation is the one in charge of typing the base case, hence $\lambda n.\, \mathsf{if}(n, \mathbf{0}, m.\, \mathbf{S}(\mathbf{S}(f\, m)))$ is typed considering that n is bound to $\mathbf{0}$, so that n is of type $\mathcal{M}$. Since π_{rec2} corresponds to a base case, there are no right premises in the rule $\texttt{t-fix}$.

We finish with the derivation of the *else* branch of the conditional:

$$\pi_{\mathrm{cond}} := \left(\dfrac{\dfrac{\dfrac{\dfrac{}{f : \{\!\{\mathcal{B}\}\!\}; \varnothing \vdash^{[]} f : \mathcal{B}}\,\texttt{t-var}_2 \qquad \dfrac{}{\varnothing; m : \mathcal{M} \vdash^{[]} m : \mathcal{M}}\,\texttt{t-var}_1}{f : \{\!\{\mathcal{B}\}\!\}; m : \mathcal{M} \vdash^{[\mathtt{B}]} f\, m : [\,]^{\mathsf{nat}}}\,\texttt{t-app}}{f : \{\!\{\mathcal{B}\}\!\}; m : \mathcal{M} \vdash^{[\mathtt{B}]} \mathbf{S}(f\, m) : [\,]^{\mathsf{nat}}}\,\texttt{t-succ}}{f : \{\!\{\mathcal{B}\}\!\}; m : \mathcal{M} \vdash^{[\mathtt{B}]} \mathbf{S}(\mathbf{S}(f\, m)) : [\,]^{\mathsf{nat}}}\,\texttt{t-succ} \right)$$

here, the term $\mathbf{S}(\mathbf{S}(f\, m))$ represents the result of doubling $\mathbf{S}(\mathbf{0})$, which is the normal form of t. As the normal form is not "used" in any way (*i.e.* there are no further interactions with the environment), it is typed with $[\,]^{\mathsf{nat}}$.

3.2 Properties of $\mathcal{H}$

This subsection develops the meta-theory of system $\mathcal{H}$ and proves the main results of this work. We begin by studying basic properties of the typing system, such as relevance and splitting lemmas. Then, we state auxiliary lemmas to prove soundness for system $\mathcal{H}$, as well as auxiliary lemmas to prove completeness.

A first remark states that a natural number must be typed with a nat-multitype:

Remark 4. If $\Phi; \Gamma \vdash^{\mathtt{m}} \mathbf{k} : \mathcal{T}$, then $\mathcal{T}$ is of the form $\mathcal{N}$.

Recall that System $\mathcal{H}$ is linear so, in particular, all assumptions are used. This property is known as *relevance*:

Lemma 5 (Relevance)**.** *If* $\pi \rhd \Phi; \Gamma \vdash^{\mathtt{m}} t : \mathcal{T}$ *then* $\mathsf{dom}(\Phi) \cup \mathsf{dom}(\Gamma) \subseteq \mathsf{fv}(t)$.

Proof. The proof is straightforward by induction on π. □

The next lemma establishes the equivalence between two different conditions regarding the typability of a value. Specifically, it states that a value is typable with a family context, a typing context, and a multi-counter that are empty if and only if it is typed with an empty ν-multiset, for any proper nature ν.

Lemma 6. *Let* $\mathbf{v}$ *be a value. Then the following are equivalent:*

1. $\Phi;\Gamma \vdash^{\mathfrak{m}} \mathbf{v} : []^{\nu}$, *where* $\nu = \mathsf{abs}$ *if* $\mathbf{v}$ *is an abstraction, and* $\nu = \mathsf{nat}$ *if* $\mathbf{v} = \mathbf{k}$
2. $\Phi = \varnothing$, $\Gamma = \varnothing$ *and* $\mathfrak{m} = []$

Proof. We prove both items by induction on $\mathbf{v}$. Cases $\mathbf{v} = \lambda x.\, t$ and $\mathbf{v} = \mathbf{0}$ are trivial. Case $\mathbf{v} = \mathbf{S}(\mathbf{l})$ is straightforward using the IH. □

The following property establishes a form of stability of the subsumption relation concerning the sum operator.

Lemma 7 (Multitype Splitting Lemma)**.** *The following hold:*

1. *If* $\bot \lhd \mathcal{T}$ *then* $\mathcal{T}$ *is of the form* $[]^{\nu}$ *for some proper nature* ν.
2. $\mathcal{T}_1^? + \mathcal{T}_2^? \lhd \mathcal{T}$ *if and only if there exist multitypes* $\mathcal{T}_1, \mathcal{T}_2$ *such that* $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2$ *and* $\mathcal{T}_i^? \lhd \mathcal{T}_i$ *for all* $i \in \{1, 2\}$.
3. $+_{i\in I}\mathcal{T}_i^? \lhd \mathcal{T}$ *if and only if either* I *is empty and* $\mathcal{T} = []^{\nu}$ *for some proper nature* ν, *or* I *is non-empty and there exist multitypes* $(\mathcal{T}_i)_{i\in I}$ *such that* $\mathcal{T} = +_{i\in I}\mathcal{T}_i$ *and* $\mathcal{T}_i^? \lhd \mathcal{T}_i$ *for all* $i \in I$.

The Multitype Splitting Lemma is used to split and merge the type derivation of values:

Lemma 8 (Generalized Value Splitting / Merging)**.** *Let* I *be a finite set,* $(\mathcal{T}_i^?)_{i\in I}$ *a family of optional multitypes and* $(\mathcal{F}_i)_{i\in I}$ *a family of multitype families. Then the following are equivalent:*

1. $\pi \rhd \Phi;\Gamma \vdash^{\mathfrak{m}} \mathbf{v} : \mathcal{T}$ *with* $\mathcal{T}$ *a multitype such that* $+_{i\in I}\mathcal{T}_i^? \lhd \mathcal{T}$
2. *There exist family contexts* $(\Phi_i)_{i\in I}$, *typing contexts* $(\Gamma_i)_{i\in I}$, *multi-counters* $(\mathfrak{m}_i)_{i\in I}$ *and multitypes* $(\mathcal{T}_i)_{i\in I}$ *such that* $\Phi = +_{i\in I}\Phi_i$ *and* $\Gamma = +_{i\in I}\Gamma_i$ *and* $\mathfrak{m} = +_{i\in I}\mathfrak{m}_i$ *and* $\pi_i \rhd \Phi_i;\Gamma_i \vdash \mathbf{v} : \mathcal{T}_i$ *and* $\mathcal{T}_i^? \lhd \mathcal{T}_i$ *for all* $i \in I$.

3.2.1 Lemmas for Soundness

To show that the type system $\mathcal{H}$ is sound, we follow well-known techniques for non-idempotent types (see [8, 2]). First, a Subject Reduction lemma is established, which is in turn based on two substitution lemmas: one for each kind of substitution that PCFH has.

Lemma 9 (Value Substitution Lemma)**.** *Let* $\zeta \rhd \Psi;\Delta \vdash^{\mathfrak{n}} \mathbf{v} : \mathcal{S}$ *and let* $\mathcal{S}^?$ *be such that* $\mathcal{S}^? \lhd \mathcal{S}$. *If* $\pi \rhd \Phi;\Gamma, x : \mathcal{S}^? \vdash^{\mathfrak{m}} t : \mathcal{T}$ *then there exists a derivation* π' *such that* $\pi' \rhd \Phi + \Psi;\Gamma + \Delta \vdash^{\mathfrak{m}+\mathfrak{n}} t\{x := \mathbf{v}\} : \mathcal{T}$.

Lemma 10 (Substitution Lemma)**.** *Let* I *be a finite set, and* $(\zeta_i \rhd \Psi_i;\Delta_i \vdash^{\mathfrak{n}_i} q : \mathcal{S}_i)_{i\in I}$ *a family of type derivations. If* $\pi \rhd \Phi, x : \{\!\{\mathcal{S}_i\}\!\}_{i\in I};\Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$ *then there exists a derivation* π' *such that* $\pi' \rhd \Phi +_{i\in I} \Psi_i;\Gamma +_{i\in I} \Delta_i \vdash^{\mathfrak{m}+_{i\in I}\mathfrak{n}_i} t\{x := q\} : \mathcal{T}$.

Both substitution lemmas are proved by induction on π.

Now we move to Subject Reduction, which gives preservation of types. Moreover, it also provides quantitative information in the sense that the multi-counter for typing t' is smaller than the multi-counter for typing t, by exactly one element.

Lemma 11 (Subject Reduction)**.** *Let t be such that $t \to_\rho t'$ and $\pi \rhd \Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$. Then there exist a derivation π' and a multi-counter $\mathfrak{m}'$ such that $[\rho] + \mathfrak{m}' = \mathfrak{m}$ and $\pi' \rhd \Phi; \Gamma \vdash^{\mathfrak{m}'} t' : \mathcal{T}$.*

Proof. By induction on the derivation of $t \to_\rho t'$. We show only cases `r-beta` and `r-fix` to see how the substitution lemmas are applied.

- `r-beta`. Then $t = (\lambda x.\, s)\, \mathbf{v} \to_{\mathtt{B}} s\{x := \mathbf{v}\} = t'$. The conclusion of π can then only be derived using rule `t-app`, so π has the form:

$$\dfrac{\dfrac{\pi_1 \rhd \Phi_1; \Gamma_1, x : \mathcal{S}^? \vdash^{\mathfrak{m}_1} s : \mathcal{T}}{\Phi_1; \Gamma_1 \vdash^{\mathfrak{m}_1} \lambda x.\, s : [\mathcal{S}^? \to \mathcal{T}]^{\mathsf{abs}}}\,\mathtt{t\text{-}abs} \quad (1)\ \mathcal{S}^? \lhd \mathcal{S} \qquad \pi_2 \rhd \Phi_2; \Gamma_2 \vdash^{\mathfrak{m}_2} \mathbf{v} : \mathcal{S}}{\Phi_1 + \Phi_2; \Gamma_1 + \Gamma_2 \vdash^{[\mathtt{B}]+\mathfrak{m}_1+\mathfrak{m}_2} (\lambda x.\, s)\, \mathbf{v} : \mathcal{T}}\,\mathtt{t\text{-}app}$$

  where $\Phi = \Phi_1 + \Phi_2$ and $\Gamma = \Gamma_1 + \Gamma_2$ and $\mathfrak{m} = [\mathtt{B}] + \mathfrak{m}_1 + \mathfrak{m}_2$. Given (1), we can apply Lemma 9 to π_1 with π_2, yielding $\pi' \rhd \Phi_1 + \Phi_2; \Gamma_1 + \Gamma_2 \vdash^{\mathfrak{m}_1+\mathfrak{m}_2} s\{x := \mathbf{v}\} : \mathcal{T}$, and we conclude with $\mathfrak{m}' = \mathfrak{m}_1 + \mathfrak{m}_2$, since $[\mathtt{B}] + \mathfrak{m}' = [\mathtt{B}] + \mathfrak{m}_1 + \mathfrak{m}_2 = \mathfrak{m}$.

- `t-fix`. Then $t = \mathsf{fix}\, x.\, s \to_\rho s\{x := \mathsf{fix}\, x.\, s\} = t'$. The conclusion of π can then only be derived by rule `t-fix`, so π has the form:

$$\dfrac{\pi_0 \rhd \Phi_0, x : \{\!\{\mathcal{S}_i\}\!\}_{i\in I}; \Gamma_0 \vdash^{\mathfrak{m}_0} s : \mathcal{T} \qquad (\pi_i \rhd \Phi_i; \Gamma_i \vdash^{\mathfrak{m}_i} \mathsf{fix}\, x.\, s : \mathcal{S}_i)_{i\in I}}{\Phi_0 +_{i\in I} \Phi_i; \Gamma_0 +_{i\in I} \Gamma_i \vdash^{[\mathtt{F}]+\mathfrak{m}_0+_{i\in I}\mathfrak{m}_i} \mathsf{fix}\, x.\, s : \mathcal{T}}\,\mathtt{t\text{-}fix}$$

  where $\Phi = \Phi_0 +_{i\in I} \Phi_i$ and $\Gamma = \Gamma_0 +_{i\in I} \Gamma_i$ and $\mathfrak{m} = \mathfrak{m}_0 +_{i\in I} \mathfrak{m}_i$. By Lemma 10 on π_0 with $(\pi_i)_{i\in I}$ we obtain $\pi' \rhd \Phi_0 +_{i\in I} \Phi_i; \Gamma_0 +_{i\in I} \Gamma_i \vdash^{\mathfrak{m}_0+_{i\in I}\mathfrak{m}_i} s\{x := \mathsf{fix}\, x.\, s\} : \mathcal{T}$. We let $\mathfrak{m}' = \mathfrak{m}_0 +_{i\in I} \mathfrak{m}_i$ and we conclude since $[\mathtt{F}] + \mathfrak{m}' = [\mathtt{F}] + \mathfrak{m}_0 +_{i\in I} \mathfrak{m}_i = \mathfrak{m}$. □

3.2.2 Lemmas for Completeness

To show that the type system $\mathcal{H}$ is complete, the procedure is analogous to the one to prove soundness. First, a Subject Expansion lemma is required, which is in turn based on two anti-substitution lemmas: one for each kind of substitution that PCFH has.

Lemma 12 (Value Anti-Substitution)**.** *Let $\pi' \rhd \Phi; \Gamma \vdash^{\mathfrak{m}} t\{x := \mathbf{v}\} : \mathcal{T}$. Then there exist family contexts Φ_1, Φ_2, typing contexts Γ_1, Γ_2, multi-counters $\mathfrak{m}_1$, $\mathfrak{m}_2$, an optional multitype $\mathcal{S}^?$ and a multitype $\mathcal{S}$ satisfying:*

1. $\pi \rhd \Phi_1; \Gamma_1, x : \mathcal{S}^? \vdash^{\mathfrak{m}_1} t : \mathcal{T}$
2. $\zeta \rhd \Phi_2; \Gamma_2 \vdash^{\mathfrak{m}_2} \mathbf{v} : \mathcal{S}$
3. $\Phi = \Phi_1 + \Phi_2$ *and* $\Gamma = \Gamma_1 + \Gamma_2$ *and* $\mathfrak{m} = \mathfrak{m}_1 + \mathfrak{m}_2$ *and* $\mathcal{S}^? \lhd \mathcal{S}$

Lemma 13 (Anti-Substitution)**.** *Let $\pi' \rhd \Phi; \Gamma \vdash^{\mathfrak{m}} t\{x := q\} : \mathcal{T}$. Then there exist a finite set I, family contexts Φ_0, $(\Psi_i)_{i\in I}$, typing contexts Γ_0, $(\Delta_i)_{i\in I}$, multi-counters $\mathfrak{m}_0$, $(\mathfrak{n}_i)_{i\in I}$ and multitypes $(\mathcal{S}_i)_{i\in I}$ satisfying:*

1. $\pi \rhd \Phi_0, x : \{\!\{\mathcal{S}_i\}\!\}_{i\in I}; \Gamma_0 \vdash^{\mathfrak{m}_0} t : \mathcal{T}$
2. $(\zeta_i \rhd \Psi_i; \Delta_i \vdash^{\mathfrak{n}_i} q : \mathcal{S}_i)_{i\in I}$
3. $\Phi = \Phi_0 +_{i\in I} \Psi_i$ *and* $\Gamma = \Gamma_0 +_{i\in I} \Delta_i$ *and* $\mathfrak{m} = \mathfrak{m}_0 +_{i\in I} \mathfrak{n}_i$

Both anti-substitution lemmas are proved by induction on π.

We can proceed now to show Subject Expansion, which is analogous to Subject Reduction in Lemma 11. However, it goes in the opposite direction of the reduction relation $t \to_\rho t'$: given a type derivation of t', it proves that t has the same type as t'.

Lemma 14 (Subject Expansion)**.** *Let $t \to_\rho t'$ and $\pi' \rhd \Phi; \Gamma \vdash^{\mathfrak{m}'} t' : \mathcal{T}$. Then there exist a derivation π and a multi-counter $\mathfrak{m}$ such that $\mathfrak{m} = [\rho] + \mathfrak{m}'$ and $\Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$.*

Furthermore, to prove completeness (*i.e.* normalization implies typability) it is necessary to show that non-stuck normal forms are typable. As stated in Section 2, the focus is only on proper normal forms since the stuck ones do not have any meaning, *i.e.* they represent computation errors that type systems such as $\mathcal{H}$ do not capture.

Lemma 15 (Normal Forms are Typable)**.** *Let $t \in \mathsf{NF}_\nu$ with ν a proper nature. Then $\Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$ for some $\mathfrak{m}, \Phi, \Gamma, \mathcal{T}$.*

We can even state a stronger result than the previous one, namely that proper normal forms can be obtained from a *tight* derivation. Moreover, the multi-counter of such tight derivations is empty, which is crucial to obtain exact bounds by tight type derivations:

Lemma 16 (Normal Forms are Tight Typable)**.** *Let $t \in \mathsf{NF}_\nu$ with ν a proper nature. Then $\varnothing; \varnothing \vdash^{[]} t : []^\nu$.*

Proof. By induction on the derivation of $t \in \mathsf{NF}_\nu$. Recall cases `nf-app`, `nf-succ-stuck` and `nf-if` do not apply since `stuck` is not a proper nature.

1. `nf-abs`. Then $t = \lambda x.\, s \in \mathsf{NF}_{\mathsf{abs}}$ and indeed $\varnothing; \varnothing \vdash^{[]} \lambda x.\, s : []^{\mathsf{abs}}$ by `t-abs`.
2. `nf-zero`. Then $t = \mathbf{0} \in \mathsf{NF}_{\mathsf{nat}}$ and indeed $\varnothing; \varnothing \vdash^{[]} \mathbf{0} : []^{\mathsf{nat}}$ by `t-zero`.
3. `nf-succ-nat`. Then $t = \mathbf{S}(s) \in \mathsf{NF}_{\mathsf{nat}}$ where $s \in \mathsf{NF}_{\mathsf{nat}}$. By IH on $s \in \mathsf{NF}_{\mathsf{nat}}$, $\varnothing; \varnothing \vdash^{[]} s : []^{\mathsf{nat}}$, hence $\varnothing; \varnothing \vdash^{[]} \mathbf{S}(s) : []^{\mathsf{nat}}$ by applying `t-succ`. □

3.2.3 Soundness and Completeness of System $\mathcal{H}$

We now turn to the main results of this paper, which are the properties of soundness and completeness of $\mathcal{H}$. Both properties give an equivalence between typability and normalization. The following theorem provides upper bounds for normalization sequences of a given term, for which no tightness condition is required:

Theorem 17 (Soundness and Completeness of System $\mathcal{H}$ with Upper Bounds)**.** *Let t be a closed term, and ν be a proper nature. The following are equivalent:*

1. $\Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$
2. *There exists a sequence of steps $t = t_0 \to_{\rho_1} t_1 \ldots \to_{\rho_n} t_n$ where $t_n \in \mathsf{NF}_\nu$ and $\#(\mathfrak{m}) \geq \#([\rho_1, \ldots, \rho_n])$.*

Proof.
(1 ⇒ 2) By induction on the size of $\mathfrak{m}$, analyzing whether $t \in \mathsf{NF}_\nu$ or not.

- If $t \in \mathsf{NF}_\nu$, then t is of the form $\lambda x.\, s$, $\mathbf{0}$, or $\mathbf{S}(\mathbf{k})$, since ν is a proper nature. Hence t can only be derived by rule `t-abs`, `t-zero`, or `t-succ` respectively. In the three cases, we can take the empty evaluation sequence and we are done.

- If $t \notin \mathsf{NF}_\nu$. Then by Proposition 2 it must exist a term t' and a rule name ρ such that $t \to_\rho t'$. By Subject Reduction there exists $\mathfrak{m}'$ such that $\mathfrak{m} = [\rho] + \mathfrak{m}'$, and $\Phi; \Gamma \vdash^{\mathfrak{m}'} t' : \mathcal{T}$. By IH on $\mathfrak{m}'$, there exists a sequence of steps $t' \to_{\rho_1} t_1 \ldots \to_{\rho_n} t_n$ where $t_n \in \mathsf{NF}_\nu$ and $\#(\mathfrak{m}') \geq \#([\rho_1, \ldots, \rho_n])$. By joining this sequence with the step $t \to_\rho t'$ we obtain the sequence $t \to_\rho t' \to_{\rho_1} t_1 \ldots \to_{\rho_n} t_n$, where $\mathfrak{m} = [\rho] + \mathfrak{m}'$, and therefore $\#(\mathfrak{m}) \geq \#([\rho, \rho_1, \ldots, \rho_n])$.

(2 ⇒ 1) By induction on n.

1. $n = 0$. Then $t \in \mathsf{NF}_\nu$, and we conclude with $\Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$ by Lemma 15.

2. $n > 0$, assuming the property holds for $n-1$. Taking the reduction sequence of $(n-1)$ steps from t_1 to t_n, and $\#(\mathfrak{m}') \geq \#([\rho_2, \ldots, \rho_n])$ we can apply the IH, yielding $\Phi; \Gamma \vdash^{\mathfrak{m}'} t_1 : \mathcal{T}$. Since $t \to_{\rho_1} t_1$, then $\Phi; \Gamma \vdash^{\mathfrak{m}} t : \mathcal{T}$ by Subject Expansion, with $\mathfrak{m} = [\rho_1] + \mathfrak{m}'$, so we conclude with $\#(\mathfrak{m}) \geq \#([\rho_1, \rho_2, \ldots, \rho_n])$. □

Soundness and completeness of system $\mathcal{H}$ restricting type derivations to tight ones provide exact bounds for normalization sequences of a given term:

Theorem 18 (Tight Soundness and Completeness of System $\mathcal{H}$)**.** *Let t be a closed term, and ν be a proper nature. The following are equivalent:*

1. $\varnothing; \varnothing \vdash^{\mathfrak{m}} t : []^\nu$ *with π a tight derivation*

2. *There exists a sequence of steps $t = t_0 \to_{\rho_1} t_1 \ldots \to_{\rho_n} t_n$ where $t_n \in \mathsf{NF}_\nu$ and $\mathfrak{m} = [\rho_1, \ldots, \rho_n]$.*

Proof. Follows the same structure as in Theorem 17, *mutatis mutandis*.

To summarize the proof, we start with soundness (1 ⇒ 2): in the case where $t \in \mathsf{NF}_\nu$, we know that t must be a value, hence the typing derivation $\varnothing; \varnothing \vdash^{\mathfrak{m}} t : []^\nu$ of the hypothesis must be such that $\mathfrak{m} = []$ by Lemma 6. To prove completeness (2 ⇒ 1), we resort to the stronger Lemma 16, rather than just to Lemma 15, to ensure that we can construct a typing derivation of the form $\varnothing; \varnothing \vdash^{[]} t : []^\nu$. □

4 Conclusions

This paper proposes a quantitative study of a hybrid evaluation strategy for PCF, called PCFH, without relying on any encoding of natural numbers or fixed-point operators.

Our key contribution is a quantitative semantics for PCFH, by means of a non-idempotent intersection type system called $\mathcal{H}$, which is sound and complete with respect to the strategy. System $\mathcal{H}$ highlights the hybrid nature of the PCFH semantics, in the sense that CBN and CBV quantitative behaviors coexist within the same framework. Moreover, not only $\mathcal{H}$ provides upper bounds for the length of evaluation sequences to normal form, but we also achieve exact bounds by refining the typing derivations of $\mathcal{H}$ to those that are *tight*.

Frameworks such as CBPV or the *Bang Calculus* are able to *encode* both CBN and CBV cohesively, by distinguishing *values* from *computations*, instead of exhibiting explicitly the hybrid behavior they have. Therefore, it would be worth studying whether PCFH can be embedded into such calculi.

An interesting question is whether the quantitative information of other hybrid settings can be expressed using type systems such as $\mathcal{H}$. Another question for future work is the study of the inhabitation problem in a hybrid-type setting, which we conjecture to be decidable, given that it was already proven to be decidable in CBN [7], CBV, and CBPV [4].

References

[1] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[2] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. *J. Funct. Program.*, 30:e14, 2020.

[3] Beniamino Accattoli and Giulio Guerrieri. Types of fireballs. In Sukyoung Ryu, editor, *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 45–66. Springer, 2018.

[4] Victor Arrial, Giulio Guerrieri, and Delia Kesner. Quantitative inhabitation for different lambda calculi in a unifying framework. *Proc. ACM Program. Lang.*, 7(POPL):1483–1513, 2023.

[5] Pablo Barenbaum, Delia Kesner, and Mariana Milicich. Hybrid intersection types for pcf (extended version). *CoRR*, abs/2404.14340, 2024.

[6] Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023.

[7] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In Josep Díaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, volume 8705 of *Lecture Notes in Computer Science*, pages 341–354. Springer, 2014.

[8] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Log. J. IGPL*, 25(4):431–464, 2017.

[9] Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul.* PhD thesis, Ecole Doctorale Physique et Sciences de la Matière (Marseille), 2007.

[10] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Log.*, 21(4):685–693, 1980.

[11] Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

[12] Gilles Dowek and Jean-Jacques Lévy. *Introduction to the Theory of Programming Languages.* Undergraduate Topics in Computer Science. Springer, 2011.

[13] Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[14] Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016.

[15] Thomas Ehrhard. A coherent differential PCF. *Log. Methods Comput. Sci.*, 19(4), 2023.

[16] Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In James Cheney and Germán Vidal, editors, *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming, Edinburgh, United Kingdom, September 5-7, 2016*, pages 174–187. ACM, 2016.

[17] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic PCF. *J. ACM*, 65(4):23:1–23:44, 2018.

[18] Philippa Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software*, pages 555–574. Springer, 1994.

[19] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[20] Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1-2):393–456, 1999.

[21] Delia Kesner and Andrés Viso. Encoding tight typing in a unified framework. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[22] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 1999.

[23] Robin Milner. Fully abstract models of typed *lambda*-calculi. *Theor. Comput. Sci.*, 4(1):1–22, 1977.

[24] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[25] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.