Observability for Pair Pattern Calculi

Antonio Bucciarelli[1], Delia Kesner[1], and Simona Ronchi Della Rocca[2]

1 **Univ Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126, CNRS, Paris, France**
2 **Dipartimento di Informatica, Università di Torino, Italy**

Abstract

Inspired by the notion of solvability in the λ-calculus, we define a notion of observability for a calculus with pattern matching. We give an intersection type system for such a calculus which is based on non-idempotent types. The typing system is shown to characterize the set of terms having canonical form, which properly contains the set of observable terms, so that typability alone is not sufficient to characterize observability. However, the inhabitation problem associated with our typing system turns out to be decidable, a result which – together with typability – allows to obtain a full characterization of observability.

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems, F.3.2 Operational Semantics, F.4.1 Proof theory

Keywords and phrases Solvability, pattern calculi, intersection types, inhabitation

Digital Object Identifier 10.4230/LIPIcs.TLCA.2015.123

1 Introduction

In these last years there has been a growing interest in *pattern λ-calculi* [16, 11, 6, 12, 10, 15] which are used to model the pattern-matching primitives of functional programming languages (*e.g.* OCAML, ML, Haskell) and proof assistants (*e.g.* Coq, Isabelle). These calculi are extensions of λ-calculus, where abstractions are written as $\lambda \mathtt{p}.\mathtt{t}$, where $\mathtt{p}$ is a *pattern* specifying the expected structure of the argument. In this paper we restrict our attention to *pair* patterns, which are expressive enough to illustrate the challenging notion of solvability/observability in the framework of pattern λ-calculi.

In order to implement different *evaluation strategies*, the use of *explicit pattern-matching* becomes appropriate, giving rise to different languages with explicit pattern-matching [6, 7, 1]. In all of them, an application $(\lambda \mathtt{p}.\mathtt{t})\mathtt{u}$ reduces to $\mathtt{t}[\mathtt{p}/\mathtt{u}]$, where $[\mathtt{p}/\mathtt{u}]$ is an explicit matching, defined by means of suitable reduction rules, which are used to decide if the argument $\mathtt{u}$ matches the pattern $\mathtt{p}$. If the matching is possible, the evaluation proceeds by computing a substitution which is applied to the body $\mathtt{t}$. Otherwise, two cases arise: either a successful matching is not possible at all, and then the term $\mathtt{t}[\mathtt{p}/\mathtt{u}]$ reduces to a *failure*, denoted by the constant `fail`, or it could become possible after the application of some pertinent substitution to the argument $\mathtt{u}$, in which case the reduction is simply *blocked*. An example of failure is caused by the term $(\lambda\langle \mathtt{z}_1, \mathtt{z}_2\rangle.\mathtt{z}_1)(\lambda \mathtt{y}.\mathtt{y})$, while a blocked reduction is caused by the term $(\lambda\langle \mathtt{z}_1, \mathtt{z}_2\rangle.\mathtt{z}_1)\mathtt{y}$.

Inspired by the notion of solvability in the λ-calculus, we define a notion of *observability* for a pair pattern calculus with explicit matching. A term $\mathtt{t}$ is said to be observable if there is a *head-context* $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}]$ reduces to a pair, which is the only data structure of the language. This notion is conservative with respect to the notion of solvability in the λ-calculus, *i.e.* $\mathtt{t}$ is solvable in the λ-calculus if and only if $\mathtt{t}$ is observable in our calculus.

Solvability in the λ-calculus is of course undecidable, but it has been characterized at least in three different ways: syntactically by the notion of head-normal form [2], operationally by the notion of head-reduction [2], and logically by an intersection type assignment system [3, 13]. The problem becomes harder when changing from the call-by-name to the call-by-value setting. Indeed, in the call-by-value λ-calculus, there are normal forms that are unsolvable, like the term $(\lambda \mathtt{z}.\Delta)(\mathtt{x}I)\Delta$, where $\Delta = \lambda \mathtt{x}.\mathtt{xx}$. The problem for the pair pattern calculus is similar to that for the call-by-value, but even harder. As in the call-by-value setting, an argument needs to be partially evaluated before being consumed. Indeed, in order to evaluate an application $(\lambda \mathtt{p}.\mathtt{t})\mathtt{u}$, it is necessary to verify if $\mathtt{u}$ matches the pattern $\mathtt{p}$, and thus the subterm $\mathtt{u}$ can be forced to be partially evaluated. However, while only discrimination between values and non-values are needed in the call-by-value setting, the possible shapes of patterns are infinite here.

The difficulty of the problem depends on two facts. First, there is no simple *syntactical* characterization of observability: indeed, we supply a notion of *canonical form* such that reducing to some canonical form is a necessary condition for being observable. But this is not sufficient: canonical forms may contain blocking explicit matchings, so that we need to know whether or not there exists a substitution being able to unblock simultaneously all these blocked forms.

This theoretical complexity is reflected in the logical characterization we supply for observability: a term $\mathtt{t}$ turns out to be observable if and only if it is typable, say with a type of the shape $\mathtt{A}_1 \to \mathtt{A}_2 \to \ldots \to \mathtt{A}_n \to \alpha$ (where α is a product type), and all the types $\mathtt{A}_i$ $(1 \leq i \leq n)$ are *inhabited.* The inhabitation problem for idempotent intersection types is known to be undecidable [17], but it has recently been proved that it is decidable in the non-idempotent case [5]. More precisely, there is a sound and complete algorithm solving the inhabitation problem of non-idempotent intersection types for the λ-calculus. In this paper, we supply a type assignment system, based on non-idempotent intersection, which assigns types to terms of our pair pattern calculus. We then extend the inhabitation algorithm given in [5] to this framework, that is substantially more complicated, due to the explicit pattern matching and the use of structural information of patterns in the typing rules. However, the paper does not only show decidability of inhabitation for the pair pattern calculus, but it *uses* the decidability result to derive a full characterization of observability, which is the main result of the paper. We thus combine typability with inhabitation in order to obtain an interesting characterization of the set of *meaningful* terms of the pair pattern calculus.

The paper is organized as follows. Sec. 2 introduces the pattern calculus. Sec. 3 presents the type system and proves a characterization of terms having canonical forms by means of typability. Sec. 4 discusses the relationship between observability and inhabitation and Sec. 5 presents a sound and complete algorithm for the inhabitation problem associated to our typing system. Sec. 6 shows a complete characterization of observability, and Sec. 7 concludes by discussing some future work.

2 The Pair Pattern Calculus

We now introduce the $\Lambda_{\mathtt{p}}$-calculus, a generalization of the λ-calculus where abstraction is extended to *patterns* and terms to pairs. Pattern matching is specified by means of an *explicit* operation. Reduction is performed only if the argument matches the abstracted pattern.

Terms and contexts of the $\Lambda_{\mathtt{p}}$-calculus are defined as follows:

$$\begin{array}{llll}
(\textbf{Patterns}) & \mathtt{p},\mathtt{q} & ::= & \mathtt{x} \mid \langle \mathtt{p},\mathtt{p}\rangle \\
(\textbf{Terms}) & \mathtt{t},\mathtt{u},\mathtt{v} & ::= & \mathtt{x} \mid \lambda\mathtt{p}.\mathtt{t} \mid \langle \mathtt{t},\mathtt{t}\rangle \mid \mathtt{tt} \mid \mathtt{t}[\mathtt{p}/\mathtt{t}] \mid \mathtt{fail} \\
(\textbf{Contexts}) & \mathtt{C} & ::= & \square \mid \lambda\mathtt{p}.\mathtt{C} \mid \langle \mathtt{C},\mathtt{t}\rangle \mid \langle \mathtt{t},\mathtt{C}\rangle \mid \mathtt{Ct} \mid \mathtt{tC}
\end{array}$$

where $\mathtt{x},\mathtt{y},\mathtt{z}$ range over a countable set of variables, and every pattern $\mathtt{p}$ is *linear*, *i.e.* every variable appears at most once in $\mathtt{p}$. We denote by $\mathtt{Id}$ the **identity function** $\lambda\mathtt{x}.\mathtt{x}$. As usual we use the abbreviation $\lambda\mathtt{p}_1\ldots\mathtt{p}_n.\mathtt{t}_1\ldots\mathtt{t}_m$ for $\lambda\mathtt{p}_1(\ldots(\lambda\mathtt{p}_n.((\mathtt{t}_1\mathtt{t}_2)\ldots\mathtt{t}_m))\ldots)$, $n \geq 0$, $m \geq 1$. Remark that every λ-term is a $\Lambda_{\mathtt{p}}$-term.

The operator $[\mathtt{p}/\mathtt{t}]$ is called an **explicit matching**. The constant $\mathtt{fail}$ denotes the failure of the matching operation. **Free** and **bound variables** of terms are defined as expected, in particular $\mathtt{fv}(\lambda\mathtt{p}.\mathtt{t}) := \mathtt{fv}(\mathtt{t}) \setminus \mathtt{fv}(\mathtt{p})$ and $\mathtt{fv}(\mathtt{t}[\mathtt{p}/\mathtt{u}]) := (\mathtt{fv}(\mathtt{t}) \setminus \mathtt{fv}(\mathtt{p})) \cup \mathtt{fv}(\mathtt{u})$. We write $\mathtt{p}\#\mathtt{q}$ iff $\mathtt{fv}(\mathtt{p})$ and $\mathtt{fv}(\mathtt{q})$ are disjoint. As usual, terms are considered modulo α-conversion. Given a context $\mathtt{C}$ and a term $\mathtt{t}$, $\mathtt{C}[\mathtt{t}]$ denotes the term obtained by replacing the unique occurrence of $\square$ in $\mathtt{C}$ by $\mathtt{t}$, allowing the capture of free variables of $\mathtt{t}$. A **head-context** is a context of the shape $(\lambda\mathtt{p}_1\ldots\mathtt{p}_n.\square)\mathtt{t}_1\ldots\mathtt{t}_m$ $(n, m \geq 0)$.

The **reduction relation** of the $\Lambda_{\mathtt{p}}$-calculus, denoted by $\rightarrow$, is the contextual closure of the following reduction rules:

$$\begin{array}{llcl@{\qquad}llcl}
(r_1) & (\lambda\mathtt{p}.\mathtt{t})\mathtt{u} & \mapsto & \mathtt{t}[\mathtt{p}/\mathtt{u}] & (r_6) & \mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\lambda\mathtt{y}.\mathtt{u}] & \mapsto & \mathtt{fail} \\
(r_2) & \mathtt{t}[\mathtt{x}/\mathtt{u}] & \mapsto & \mathtt{t}\{\mathtt{x}/\mathtt{u}\} & (r_7) & \mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{fail}] & \mapsto & \mathtt{fail} \\
(r_3) & \mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\langle\mathtt{u}_1,\mathtt{u}_2\rangle] & \mapsto & \mathtt{t}[\mathtt{p}_1/\mathtt{u}_1][\mathtt{p}_2/\mathtt{u}_2] & (r_8) & \mathtt{fail}\ \mathtt{t} & \mapsto & \mathtt{fail} \\
(r_4) & \mathtt{t}[\mathtt{p}/\mathtt{v}]\mathtt{u} & \mapsto & (\mathtt{tu})[\mathtt{p}/\mathtt{v}] & (r_9) & \mathtt{fail}[\mathtt{p}/\mathtt{t}] & \mapsto & \mathtt{fail} \\
(r_5) & \mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{u}[\mathtt{q}/\mathtt{v}]] & \mapsto & \mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{u}][\mathtt{q}/\mathtt{v}] & (r_{10}) & \lambda\mathtt{p}.\mathtt{fail} & \mapsto & \mathtt{fail} \\
 & & & & (r_{11}) & \langle\mathtt{t},\mathtt{u}\rangle\mathtt{v} & \mapsto & \mathtt{fail}
\end{array}$$

where $\mathtt{t}\{\mathtt{x}/\mathtt{u}\}$ denotes the substitution of all the free occurrences of $\mathtt{x}$ in $\mathtt{t}$ by $\mathtt{u}$. By α-conversion, and without loss of generality, no reduction rule captures free variables. Thus for example in rule r_4 the bound and free variables of the term $\mathtt{t}[\mathtt{p}/\mathtt{v}]\mathtt{u}$ are supposed to be disjoint, so that the variables of $\mathtt{p}$ (which are bound in the whole term) cannot be free in $\mathtt{u}$. The reflexive and transitive closure of $\rightarrow$ is written $\rightarrow^*$.

The rule (r_1) triggers the pattern operation while rule (r_2) performs substitution, rules (r_3), (r_6) and (r_7) implement (successful or unsuccessful) pattern matching. Rules (r_8), (r_9) and (r_{10}) deal with propagation of failure. Rules (r_4) and (r_5) may seem unnecessary, and the calculus would be also confluent without them, but they are particularly useful for the design of the inhabitation algorithm (see Sec. 5). Indeed, rule (r_4) pushes *head* explicit matchings out, and rule (r_5) eliminates *nested* explicit matchings, *i.e.* matchings of the form $\mathtt{t}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{u}[\mathtt{q}/\mathtt{v}]]$. Notice that confluence would be lost if we allow (r_5) on the more general form: $\mathtt{t}[\mathtt{p}/\mathtt{u}[\mathtt{q}/\mathtt{v}]] \mapsto \mathtt{t}[\mathtt{p}/\mathtt{u}][\mathtt{q}/\mathtt{v}]$. Indeed, the following critical pair could not be closed: $\mathtt{y}[\langle\mathtt{z}_1,\mathtt{z}_2\rangle/\mathtt{z}]\ {}_{r_5,r_2}^{\ \ *}\!\!\leftarrow \mathtt{y}[\mathtt{x}/\mathtt{u}[\langle\mathtt{z}_1,\mathtt{z}_2\rangle/\mathtt{z}]] \rightarrow_{r_2} \mathtt{y}$.

▶ **Lemma 1.**
1. *The reduction relation $\rightarrow$ is confluent.*
2. *Every infinite $\rightarrow$-reduction sequence contains an infinite number of $\rightarrow_{r_2}$-reduction steps.*

The proof of the first item relies on the decreasing diagram technique [18]; that of the second one is by induction on a suitable syntactic measure.

Canonical forms are terms defined by the following grammar:

$$\mathcal{J} ::= \lambda\mathtt{p}.\mathcal{J} \mid \langle\mathtt{t},\mathtt{t}\rangle \mid \mathcal{K} \mid \mathcal{J}[\langle\mathtt{p},\mathtt{q}\rangle/\mathcal{K}] \qquad\qquad \mathcal{K} ::= \mathtt{x} \mid \mathcal{K}\mathtt{t}$$

A term $\mathtt{t}$ **is in canonical form** (or it is **canonical**), written cf, if it is generated by $\mathcal{J}$, and it **has a canonical form** if it reduces to a term in cf. Note that the cf of a term is not unique, *e.g.* both $\langle \mathtt{Id}, \mathtt{Id}\ \mathtt{Id}\rangle$ and $\langle \mathtt{Id}, \mathtt{Id}\rangle$ are cfs of $(\lambda \mathtt{xy}.\langle \mathtt{x}, \mathtt{y}\rangle)\ \mathtt{Id}\ (\mathtt{Id}\ \mathtt{Id})$. It is worth noticing that cfs and normal forms do not coincide. For example, the terms $\lambda\langle \mathtt{x}, \mathtt{y}\rangle.(\mathtt{x}(\Delta\Delta))[\langle \mathtt{z}_1, \mathtt{z}_2\rangle/\mathtt{y}\mathtt{Id}]$ and $\langle \mathtt{Id}, \mathtt{Id}\ \mathtt{Id}\rangle$ are in cf, but not in normal form, while $\mathtt{fail}$ is in normal form but not in cf. Every head normal-form in the λ-calculus is a cf in the $\Lambda_{\mathtt{p}}$-calculus.

On the pathway towards the definition of an adequate notion of *solvability* for the $\Lambda_{\mathtt{p}}$-calculus, we first recall the notion of solvability for the λ-calculus. A term $\mathtt{t}$ is solvable iff there is a head-context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}]$ reduces to $\mathtt{Id}$. It is clear that pairs have to be taken into account in order to extend the notion of solvability to the pair pattern calculus. When should a pair be considered as meaningful? At least two choices are possible: the *lazy* semantics considers a pair as meaningful in itself, the *strict* one requires both of its components to be meaningful. The first choice is adopted in this paper, since being a pair is already an observable property, particularly sufficient to unblock an explicit matching, independently from the observability of its components.

Thus, a term $\mathtt{t}$ is said to be **observable** iff there is a head-context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}]$ reduces to a pair, *i.e.* $\mathtt{C}[\mathtt{t}] \rightarrow^* \langle \mathtt{t}_1, \mathtt{t}_2\rangle$, for some terms $\mathtt{t}_1, \mathtt{t}_2 \in \Lambda_{\mathtt{p}}$. Thus for example, the term $\langle \Delta\Delta, \Delta\Delta\rangle$, consisting of a pair of unsolvable terms $\Delta\Delta$, is observable. This notion of observability turns out to be conservative with respect to that of solvability for the λ-calculus (see Theorem 23).

3 The Type System $\mathcal{P}$

In this section we present a type system for the $\Lambda_{\mathtt{p}}$-calculus, and we show that it characterizes terms having canonical form.

The set $\mathcal{T}$ of types is generated by the following grammar:

$$\begin{array}{llll}
\alpha & ::= & o \mid \times_1(\tau) \mid \times_2(\tau) & \text{(product types)} \\
\sigma, \tau, \pi, \rho & ::= & \alpha \mid \mathtt{A} \to \sigma & \text{(strict types)} \\
\mathtt{B} & ::= & [\sigma_i]_{i \in I}\ (I \neq \emptyset) & \text{(non-empty multiset types)} \\
\mathtt{A} & ::= & [\,] \mid \mathtt{B} & \text{(multiset types)}
\end{array}$$

where I is a finite set of indices. The arrow constructor is right associative. We consider a unique type constant o, which can be assigned to any pair.

We write $\mathtt{supp}(\mathtt{A})$ to denote the *support set* of the multiset $\mathtt{A}$, $\sqcup$ for multiset union and $\in$ to denote multiset membership. The **product** operation $\mathbb{X}$ on multisets is defined as follows:

$$\begin{array}{ccccll}
[\,] & \mathbb{X} & [\,] & := & [o] & \\
[\sigma_i]_{i\in I} & \mathbb{X} & [\rho_j]_{j\in J} & := & [\times_1(\sigma_i)]_{i\in I} \sqcup [\times_2(\rho_j)]_{j\in J} & \text{if } I \neq \emptyset \text{ or } J \neq \emptyset
\end{array}$$

Remark that $\sqcup_{i\in I}\mathtt{A}_i \mathbb{X} \sqcup_{i\in I} \mathtt{A}'_i \sqsubseteq \sqcup_{i\in I}(\mathtt{A}_i \mathbb{X} \mathtt{A}'_i)$, the multiset inclusion being strict for example in the following case: $([\,] \sqcup [\,])\mathbb{X}([\,] \sqcup [\,]) = [o] \sqsubset [o, o] = ([\,]\mathbb{X}[\,]) \sqcup ([\,]\mathbb{X}[\,])$.

The **structure** of a pattern describes its shape, it is defined as follows:

$$\begin{array}{ccc}
\mathcal{S}(\mathtt{x}) & := & [\,] \\
\mathcal{S}(\langle \mathtt{p}_1, \mathtt{p}_2\rangle) & := & \mathcal{S}(\mathtt{p}_1)\mathbb{X}\mathcal{S}(\mathtt{p}_2)
\end{array}$$

E.g. $\mathcal{S}(\langle \mathtt{x}, \mathtt{y}\rangle) = [o]$, $\mathcal{S}(\langle \mathtt{x}, \langle \mathtt{y}, \mathtt{z}\rangle\rangle) = [\times_2(o)]$ and $\mathcal{S}(\langle\langle \mathtt{x}, \mathtt{w}\rangle, \langle \mathtt{y}, \mathtt{z}\rangle\rangle) = [\times_1(o), \times_2(o)]$. Notice that $\mathcal{S}(\mathtt{p})$ is nothing but a description of $\mathtt{p}$ seen as a binary tree whose leaves are distinct variables, and whose nodes are labeled by the pair constructor. Indeed, each element of $\mathcal{S}(\mathtt{p})$ specifies a maximal branch of such a tree, *i.e.* a branch whose last node is a pair constructor, and whose children are both leaves (*i.e.* variables). $\mathcal{S}(\mathtt{p})$ should be understood as the multiset

$$\frac{}{\mathtt{x}:\mathtt{B}\Vdash \mathtt{x}:\mathtt{B}}\ (\mathtt{varpat}) \qquad \frac{}{\Vdash \mathtt{x}:[\,]}\ (\mathtt{weakpat}) \qquad \frac{\Gamma\Vdash \mathtt{p}:\mathtt{A}_1 \quad \Delta\Vdash \mathtt{q}:\mathtt{A}_2 \quad \mathtt{p}\#\mathtt{q}}{\Gamma+\Delta\Vdash \langle \mathtt{p},\mathtt{q}\rangle:\mathtt{A}_1\mathbb{X}\mathtt{A}_2}\ (\mathtt{pairpat})$$

$$\frac{}{\mathtt{x}:[\pi]\vdash \mathtt{x}:\pi}\ (\mathtt{var}) \qquad \frac{\Gamma\vdash \mathtt{t}:\pi \quad \Gamma|_{\mathtt{p}}\Vdash \mathtt{p}:[\sigma_i]_{i\in I} \quad (\sigma_j\in\mathcal{S}(\mathtt{p}))_{j\in J} \quad I\cap J=\emptyset}{\Gamma\setminus\Gamma|_{\mathtt{p}}\vdash \lambda\mathtt{p}.\mathtt{t}:[\sigma_k]_{k\in I\cup J}\to\pi}\ (\to\mathtt{i})$$

$$\frac{\Gamma\vdash \mathtt{t}:[\sigma_i]_{i\in I}\to\pi \quad (\Delta_i\vdash \mathtt{u}:\sigma_i)_{i\in I}}{\Gamma+_{i\in I}\Delta_i\vdash \mathtt{tu}:\pi}\ (\to\mathtt{e})$$

$$\frac{}{\vdash\langle \mathtt{t},\mathtt{u}\rangle:o}\ (\mathtt{emptypair}) \qquad \frac{\Gamma\vdash \mathtt{t}:\sigma}{\Gamma\vdash\langle \mathtt{t},\mathtt{u}\rangle:\times_1(\sigma)}\ (\mathtt{pair1}) \qquad \frac{\Gamma\vdash \mathtt{u}:\tau}{\Gamma\vdash\langle \mathtt{t},\mathtt{u}\rangle:\times_2(\tau)}\ (\mathtt{pair2})$$

$$\frac{\Gamma\vdash \mathtt{t}:\sigma \quad \Gamma|_{\mathtt{p}}\Vdash \mathtt{p}:[\sigma_i]_{i\in I} \quad (\sigma_j\in\mathcal{S}(\mathtt{p}))_{j\in J} \quad (\Delta_k\vdash \mathtt{u}:\sigma_k)_{k\in I\cup J} \quad I\cap J=\emptyset}{(\Gamma\setminus\Gamma|_{\mathtt{p}})+_{k\in I\cup J}\Delta_k\vdash \mathtt{t}[\mathtt{p}/\mathtt{u}]:\sigma}\ (\mathtt{sub})$$

Figure 1 The type assignment system $\mathcal{P}$.

of *non depletable* resources associated with $\mathtt{p}$; the persistent character of these resources is highlighted in the forthcoming typing system.

Typing environments, written Γ,Δ, are functions from variables to multiset types, assigning the empty multiset to almost all the variables. The **domain** of Γ, written $\mathtt{dom}(\Gamma)$, is the set of variables whose image is different from $[\,]$. We write $\Gamma\#\Delta$ iff $\mathtt{dom}(\Gamma)\cap\mathtt{dom}(\Delta)=\emptyset$.

▶ Notation 2. Given the environments $\{\Gamma_i\}_{i\in I}$, we write $+_{i\in I}\Gamma_i$ for the environment which maps $\mathtt{x}$ to $\sqcup_{i\in I}\Gamma_i(\mathtt{x})$. If $I=\emptyset$, the resulting environment is the one having an empty domain. Note that $\Gamma+\Delta$ and $\Gamma+_{i\in I}\Delta_i$ are just particular cases of the previous general definition. When $\Gamma\#\Delta$ we write $\Gamma;\Delta$ instead of $\Gamma+\Delta$. We write $\Gamma\setminus\mathtt{x}$ for the environment assigning $[\,]$ to $\mathtt{x}$, and acting as Γ otherwise; $\mathtt{x}_1:\mathtt{A}_1;\ldots;\mathtt{x}_n:\mathtt{A}_n$ is the environment assigning $\mathtt{A}_i$ to $\mathtt{x}_i$, for $1\leq i\leq n$, and $[\,]$ to any other variable; $\Gamma|_{\mathtt{p}}$ denotes the environment such that $\Gamma|_{\mathtt{p}}(\mathtt{x})=\Gamma(\mathtt{x})$, if $\mathtt{x}\in\mathtt{fv}(\mathtt{p})$, $[\,]$ otherwise.

The **type assignment system** $\mathcal{P}$ (see Fig. 1) is a set of typing rules assigning strict types of $\mathcal{T}$ to terms of $\Lambda_{\mathtt{p}}$. We write $\Pi\triangleright\Gamma\vdash\mathtt{t}:\sigma$ (resp. $\Pi\triangleright\Gamma\Vdash\mathtt{p}:\mathtt{A}$) to denote a **typing derivation** ending in the sequent $\Gamma\vdash\mathtt{t}:\sigma$ (resp. $\Gamma\Vdash\mathtt{p}:\mathtt{A}$), in which case $\mathtt{t}$ (resp. $\mathtt{p}$) is called the **subject** of Π; by abuse of notation, $\Gamma\vdash\mathtt{t}:\sigma$ (resp. $\Gamma\Vdash\mathtt{p}:\mathtt{A}$) also denotes the existence of some typing derivation ending in this sequent, in which case $\mathtt{t}$ (resp. $\mathtt{p}$) is said to be **typable**. The **measure** of a typing derivation Π, written $\mathtt{meas}(\Pi)$, is the number of typing rules in Π.

Rules $(\mathtt{var})$ and $(\to\mathtt{e})$ are those used for λ-calculus in [5, 8]. Linearity of patterns is guaranteed by the clause $\mathtt{p}\#\mathtt{q}$ in rule $(\mathtt{pairpat})$. Rule $(\mathtt{weakpat})$ is essential to type *erasing* functions such as for example $\lambda\mathtt{x}.\mathtt{Id}$. The rule $(\mathtt{emptypair})$ types for example $\langle\Delta\Delta,\Delta\Delta\rangle$, and thus $(\lambda\langle\mathtt{x},\mathtt{y}\rangle.\mathtt{Id})\langle\Delta\Delta,\Delta\Delta\rangle$. Rules $(\mathtt{pair1})$ and $(\mathtt{pair2})$ type pairs having just one typed component, whereas standard typed calculi with pairs (e.g. [6]) requires both components to be typed. This is necessary to type terms like $(\lambda\langle\mathtt{x},\mathtt{y}\rangle.\mathtt{x})\langle\mathtt{Id},\Delta\Delta\rangle$. Moreover, the standard policy can be easily recovered from ours by typing a pair whose components are both typed using $(\mathtt{pair1})$ and $(\mathtt{pair2})$ successively.

The rules $(\to \mathtt{i})$ and $(\mathtt{sub})$ are the most subtle ones [1]. Here is where the structural types come into play: they can be used *ad libitum* (whence the notation $(\sigma_j \in \mathcal{S}(\mathtt{p}))_{j\in J}$), thanks to *non depletable* nature of the information provided by the structure of patterns (whereas the type information of variables should be understood as *depletable*). Concerning more specifically the rule $(\mathtt{sub})$: in order to type $\mathtt{t}[\mathtt{p}/\mathtt{u}]$, on one hand we need to type $\mathtt{t}$ and on the other one we need to check that $\mathtt{p}$ and $\mathtt{u}$ can be assigned the same types. Since the system is relevant, we need to collect the environments used in all the premises typing $\mathtt{p}$ and $\mathtt{u}$. Remark however that there is a lack of symmetry between patterns and terms: while the only information we can use about terms is the one concerning their types, a pattern $\mathtt{p}$ has not only a type (description of its depletable resources), but also an intrinsic shape that is completely described by the structural (non depletable) types in the set $\mathcal{S}(\mathtt{p})$.

Actually the structural information on patterns is necessary, in particular, to guarantee subject reduction for rule (r_5). Indeed, given $\mathtt{t} = \lambda\mathtt{w}.(\mathtt{zz'})[\langle\mathtt{z},\mathtt{z'}\rangle/(\mathtt{yx})[\langle\mathtt{x},\mathtt{x'}\rangle/\mathtt{w}]] \to_{r_5} \lambda\mathtt{w}.(\mathtt{zz'})[\langle\mathtt{z},\mathtt{z'}\rangle/(\mathtt{yx})][\langle\mathtt{x},\mathtt{x'}\rangle/\mathtt{w}] = \mathtt{t'}$, and $\Gamma = \mathtt{y} : [[\,] \to \times_1(\tau), [\pi] \to \times_2(\sigma)]$, we have that $\Gamma \vdash \mathtt{t} : [o, \times_1(\pi)] \to \sigma$, but $\Gamma \vdash \mathtt{t'} : [o, \times_1(\pi)] \to \sigma$ holds only by using the fact that $o \in \mathcal{S}(\langle\mathtt{x},\mathtt{x'}\rangle)$. This counterexample shows that a clear tension appears between the rewriting rule (r_5) and the use of the structural set $\mathcal{S}(\mathtt{p})$ in the typing rules $(\to \mathtt{i})$ and $(\mathtt{sub})$. Eliminating (r_5) from the reduction system would certainly simplify the typing system, but would significantly complicate the inhabitation algorithm that will be presented in Sec. 5.

▶ **Example 3.** The following (partially described) derivation is valid:

$$\dfrac{\begin{array}{c}(a)\ \mathtt{x} : [\alpha] \vdash \mathtt{x} : \alpha \qquad (b)\ \mathtt{x} : [\alpha] \Vdash \langle\mathtt{x},\mathtt{y}\rangle : [\times_1(\alpha)] \\ (c)\ (o \in \mathcal{S}(\langle\mathtt{x},\mathtt{y}\rangle)) \qquad (d)\ \mathtt{z} : [o] \vdash \mathtt{z} : o \qquad (e)\ \mathtt{z} : [\times_1(\alpha)] \vdash \mathtt{z} : \times_1(\alpha)\end{array}}{\mathtt{z} : [o, \times_1(\alpha)] \vdash \mathtt{x}[\langle\mathtt{x},\mathtt{y}\rangle/\mathtt{z}] : \alpha}\ (\mathtt{sub})$$

Using only the hypothesis (a), (b) and (e) we get another valid typing derivation ending in $\mathtt{z} : [\times_1(\alpha)] \vdash \mathtt{x}[\langle\mathtt{x},\mathtt{y}\rangle/\mathtt{z}] : \alpha$ which does not use structural information about the pattern $\langle\mathtt{x},\mathtt{y}\rangle$.

The system is relevant, in the sense that only the used premises are registered in the typing environments. This property, formally stated in the following lemma, will be an important technical tool used to develop the inhabitation algorithm.

▶ **Lemma 4** (Relevance).
- *If* $\Gamma \Vdash \mathtt{p} : \mathtt{A}$, *then* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(\mathtt{p})$.
- *If* $\Gamma \vdash \mathtt{t} : \sigma$, *then* $\mathtt{dom}(\Gamma) \subseteq \mathtt{fv}(\mathtt{t})$.

Proof. By induction on the typing derivations. ◀

Some useful properties will be needed in the sequel. In particular, the next technical lemma says that, given different types $\mathtt{A}_i$ for a given pattern $\mathtt{p}$, it is always possible to split $\sqcup_{i\in I}\mathtt{A}_i$ into a bunch of resource types $\mathtt{A}$ and another one of structural types $\mathtt{A'}$.

▶ **Lemma 5.** *Let* $I \neq \emptyset$. *If* $(\Gamma_i \Vdash \mathtt{p} : \mathtt{A}_i)_{i\in I}$, *then there exist* $\mathtt{A}, \mathtt{A'}$ *such that*
1. $\mathtt{A} \sqcup \mathtt{A'} = \sqcup_{i\in I}\mathtt{A}_i$,
2. $+_{i\in I}\Gamma_i \Vdash \mathtt{p} : \mathtt{A}$

[1] Notice that "Γ", "$\Gamma|_p$" and "$\Gamma \setminus \Gamma|_p$" in rules $(\to \mathtt{i})$ and $(\mathtt{sub})$ could be replaced by "$\Gamma_1;\Gamma_2$", "Γ_2" and "Γ_1", respectively, only if $\mathtt{dom}(\Gamma_1) \cap \mathtt{fv}(\mathtt{p}) = \emptyset$. Otherwise, for instance, $\lambda\mathtt{x}.\mathtt{x}$ would be typable with type $[\,] \to \sigma$.

3. $\mathtt{A} = [\,]$ *implies* $\mathtt{A}' = [\,]$,
4. $\mathtt{supp}(\mathtt{A}') \subseteq \mathcal{S}(\mathtt{p})$,
5. $\mathtt{meas}(+_{i\in I}\Gamma_i \Vdash \mathtt{p} : \mathtt{A}) \leq \Sigma_{i\in I}\mathtt{meas}(\Gamma_i \Vdash \mathtt{p} : \mathtt{A}_i)$.

Proof. By induction on $\mathtt{p}$. ◀

The following lemma can be shown by induction on typing derivations; it is used in the forthcoming subject reduction property.

▶ **Lemma 6** (Substitution Lemma). *If* $\Pi \triangleright \Gamma; \mathtt{x} : [\rho_i]_{i\in I} \vdash \mathtt{t} : \tau$, *and* $(\Theta_i \triangleright \Delta_i \vdash \mathtt{u} : \rho_i)_{i\in I}$ *then* $\Pi' \triangleright \Gamma +_{i\in I} \Delta_i \vdash \mathtt{t}\{\mathtt{x}/\mathtt{u}\} : \tau$ *where* $\mathtt{meas}(\Pi') < \mathtt{meas}(\Pi) + \sum_{i\in I} \mathtt{meas}(\Theta_i)$.

Notice that, in the process of assigning a type to a term $\mathtt{t}$, some subterms of $\mathtt{t}$ may be left untyped. Typically, this happens when $\mathtt{t}$ contains occurrences of non typable terms, like in $\lambda\mathtt{x}.\mathtt{x}(\Delta\Delta)$. We are then going to define the notion of **typed occurrence** of a typing derivation, which plays an essential role in the rest of this paper: indeed, thanks to the use of non-idempotent intersection types, a combinatorial argument based on a measure on typing derivations (*cf.* Lem. 9.1), allows to prove the termination of reduction of redexes occurring in typed occurrences of their respective typing derivations.

Let us then define an **occurrence** of a subterm $\mathtt{u}$ in a term $\mathtt{t}$ as a context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{u}] = \mathtt{t}$. Then, given a typing derivation $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$, an occurrence of a subterm of $\mathtt{t}$ is a typed occurrence of Π if and only if it is the subject of a subderivation of Π. More precisely:

▶ **Definition 7.** Given a type derivation Π, the set of **typed occurrences** of Π, written $\mathtt{toc}(\Pi)$, by induction on the last rule of Π.

- If Π ends with $(\mathtt{var})$, then $\mathtt{toc}(\Pi) := \{\Box\}$.
- If Π ends with $(\mathtt{pair1})$ with subject $\langle\mathtt{u},\mathtt{v}\rangle$ and premise Π', then $\mathtt{toc}(\Pi) := \{\Box\} \cup \{\langle\mathtt{C},\mathtt{v}\rangle \mid \mathtt{C} \in \mathtt{toc}(\Pi')\}$.
- If Π ends with $(\mathtt{pair2})$ with subject $\langle\mathtt{u},\mathtt{v}\rangle$ and premise Π' then $\mathtt{toc}(\Pi) := \{\Box\} \cup \{\langle\mathtt{u},\mathtt{C}\rangle \mid \mathtt{C} \in \mathtt{toc}(\Pi')\}$.
- If Π ends with $(\rightarrow \mathtt{i})$ with subject $\lambda\mathtt{p}.\mathtt{u}$ and premise Π' then $\mathtt{toc}(\Pi) := \{\Box\} \cup \{\lambda\mathtt{p}.\mathtt{C} \mid \mathtt{C} \in \mathtt{toc}(\Pi')\}$.
- If Π ends with $(\rightarrow \mathtt{e})$ with subject $\mathtt{tu}$ and premises Π_1 and Π_k $(k \in K)$ with subjects $\mathtt{t}$ and $\mathtt{u}$ respectively, then $\mathtt{toc}(\Pi) := \{\Box\} \cup \{\mathtt{tC} \mid \mathtt{C} \in \mathtt{toc}(\Pi_k), k \in K\} \cup \{\mathtt{Cu} \mid \mathtt{C} \in \mathtt{toc}(\Pi_1)\}$.
- If Π ends with $(\mathtt{sub})$ with subject $\mathtt{t}[\mathtt{p}/\mathtt{u}]$ and premises Π_1 and Π_k $(k \in K)$ with subjects $\mathtt{t}$ and $\mathtt{u}$ respectively, then $\mathtt{toc}(\Pi) := \{\Box\} \cup \{\mathtt{C}[\mathtt{p}/\mathtt{u}] \mid \mathtt{C} \in \mathtt{toc}(\Pi_1)\} \cup \{\mathtt{t}[\mathtt{p}/\mathtt{C}] \mid \mathtt{C} \in \mathtt{toc}(\Pi_k), k \in K\}$.

▶ **Example 8.** Given the following derivations Π and Π', the occurrences $\Box$ and $\Box\mathtt{y}$ belong to both $\mathtt{toc}(\Pi)$ and $\mathtt{toc}(\Pi')$ while $\mathtt{x}\Box$ belongs to $\mathtt{toc}(\Pi)$ but not to $\mathtt{toc}(\Pi')$.

$$\Pi \triangleright \frac{\mathtt{x} : [[\tau] \to \tau] \vdash \mathtt{x} : [\tau] \to \tau \qquad \mathtt{y} : [\tau] \vdash \mathtt{y} : \tau}{\mathtt{x} : [[\tau] \to \tau], \mathtt{y} : [\tau] \vdash \mathtt{xy} : \tau} \qquad \Pi' \triangleright \frac{\mathtt{x} : [[\,] \to \tau] \vdash \mathtt{x} : [\,] \to \tau}{\mathtt{x} : [[\,] \to \tau] \vdash \mathtt{xy} : \tau}$$

Given $\Pi \triangleright \Gamma \vdash \mathtt{t} : \tau$, $\mathtt{t}$ is said to be in **Π-normal form**, also written **Π-nf**, if for every typed occurrence $\mathtt{C} \in \mathtt{toc}(\Pi)$ such that $\mathtt{t} = \mathtt{C}[\mathtt{u}]$, the subterm $\mathtt{u}$ is not a redex.

The system $\mathcal{P}$ enjoys both subject reduction and subject expansion. In particular, thanks to the use of multisets, subject reduction decreases the measure of the derivation, in case a substitution is performed by rule (r_2) and the redex is typed. This property allows for a simple proof of the "only if" part of the characterization theorem.

▶ **Lemma 9.**

1. **(Weighted Subject Reduction)** *If* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \tau$ *and* $\mathtt{t} \to \mathtt{v}$*, then* $\Pi' \triangleright \Gamma \vdash \mathtt{v} : \tau$ *and* $\mathtt{meas}(\Pi') \leq \mathtt{meas}(\Pi)$*. Moreover, if the reduced redex is* (r_2) *and it occurs in a typed occurrence of* Π*, then* $\mathtt{meas}(\Pi') < \mathtt{meas}(\Pi)$*.*
2. **(Subject Expansion)** *If* $\Gamma \vdash \mathtt{v} : \sigma$ *and* $\mathtt{t} \to \mathtt{v}$*, then* $\Gamma \vdash \mathtt{t} : \sigma$*.*

Proof. 1. By induction on $\mathtt{t} \to \mathtt{v}$ using Lemmas 5, 6 and 4.
2. By induction on $\mathtt{t} \to \mathtt{v}$.

◀

We are now ready to provide the logical characterization of terms having canonical form.

▶ **Theorem 10** (Characterization). *A term* $\mathtt{t}$ *is typable iff* $\mathtt{t}$ *has a canonical form.*

Proof.
- (if) We reason by induction on the grammar defining the canonical forms. We first prove that for all type σ and for all $\mathcal{K}$-canonical form $\mathtt{t}$, $\mathtt{t}$ can be typed by σ. In fact every $\mathcal{K}$-canonical form is of the shape $\mathtt{x}\mathtt{t}_1 ... \mathtt{t}_n$, for $n \geq 0$. It is easy to check that $\mathtt{x} : \underbrace{[\,] \to ... \to [\,]}_{n} \to \sigma \vdash \mathtt{x}\mathtt{t}_1 ... \mathtt{t}_n : \sigma$. Let $\mathtt{t}$ be a $\mathcal{J}$-canonical form. If $\mathtt{t} = \langle \mathtt{u}, \mathtt{v} \rangle$ then by rule $(\mathtt{emptypair}) \vdash \langle \mathtt{u}, \mathtt{v} \rangle : o$. If $\mathtt{t} = \lambda \mathtt{p}.\mathtt{u}$, then by induction $\mathtt{u}$ can be typed and the result follows from rule $(\to \mathtt{I})$. Let $\mathtt{t} = \mathtt{t}'[\langle \mathtt{p}, \mathtt{q} \rangle / \mathtt{v}]$, where $\mathtt{t}'$ (resp. $\mathtt{v}$) is a $\mathcal{J}$ (resp. $\mathcal{K}$) canonical form. By the *i.h.* there are Γ, σ such that $\Gamma \vdash \mathtt{t}' : \sigma$. Moreover, it is easy to see that $\Gamma|_{\langle \mathtt{p}, \mathtt{q} \rangle} \Vdash \langle \mathtt{p}, \mathtt{q} \rangle : [\sigma_i]_{i \in I}$, for some $[\sigma_i]_{i \in I}$. Since $\mathtt{v}$ is a $\mathcal{K}$-canonical form, then $\Delta_i \vdash \mathtt{v} : \sigma_i$ for all $i \in I$, as shown above. Thus $\Gamma +_{i \in I} \Delta_i \vdash \mathtt{t}'[\langle \mathtt{p}, \mathtt{q} \rangle / \mathtt{v}] : \sigma$ by rule $(\mathtt{sub})$ with $J = \emptyset$.
- (only if) Let $\mathtt{t}$ be a typable term, *i.e.* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$. Consider a reduction strategy $\mathcal{ST}$ that always chooses a *typed* redex occurrence. By Lem. 9.1 and Lem. 1.2 the strategy $\mathcal{ST}$ always terminates. Let $\mathtt{t}'$ be a normal-form of $\mathtt{t}$ for the strategy $\mathcal{ST}$, *i.e.* $\mathtt{t}$ reduces to $\mathtt{t}'$ using $\mathcal{ST}$, and $\mathcal{ST}$ applied to $\mathtt{t}'$ is undefined. We know that $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ by Lem. 9.1. Then, by definition of $\mathcal{ST}$, $\mathtt{t}'$ has no typed redex occurrence. A simple induction on $\mathtt{t}'$ allows to conclude that it is a canonical form. ◀

4 From canonicity to observability

We proved in the previous section that system $\mathcal{P}$ gives a complete characterization of terms having canonical forms. The next theorem proves that system $\mathcal{P}$ is complete with respect to observability.

▶ **Theorem 11.** *Observability implies typability.*

Proof. If $\mathtt{t}$ is observable, then there is a head context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}]$ reduces to $\langle \mathtt{u}, \mathtt{v} \rangle$, for some $\mathtt{u}$ and $\mathtt{v}$. Since all pairs are typable, the term $\mathtt{C}[\mathtt{t}]$ is typable by Lem. 9.2. Remember that $\mathtt{C}[\mathtt{t}] = (\lambda \mathtt{p}_1 \ldots \mathtt{p}_n.\mathtt{t})\mathtt{t}_1 ... \mathtt{t}_m$ so that $\mathtt{t}$ is typable too, by easy inspection of the typing system. ◀

Unfortunately, soundness does not hold, *i.e.* the set of observable terms is strictly included in the set of terms having canonical form, as shown below.

▶ **Example 12.** The term $\mathtt{t}_1 = \lambda \mathtt{x}.\mathtt{Id}[\langle \mathtt{y}, \mathtt{z} \rangle / \mathtt{x}][\langle \mathtt{y}', \mathtt{z}' \rangle / \mathtt{x}\mathtt{Id}]$ is canonical, hence typable (by Thm. 10), but not observable. In fact, it is easy to see that there is no term $\mathtt{u}$ such that both $\mathtt{u}$ and $\mathtt{u}\mathtt{Id}$ reduce to pairs. A less trivial example is the term $\mathtt{t}_2 = \lambda \mathtt{x}.\mathtt{Id}[\langle \mathtt{y}, \mathtt{z} \rangle / \mathtt{x}\langle \mathtt{Id}, \mathtt{Id} \rangle][\langle \mathtt{y}', \mathtt{z}' \rangle / \mathtt{x}\mathtt{Id}\mathtt{Id}]$, which is canonical, hence typable, but not observable, as proved in the next lemma.

▶ **Lemma 13.** *There is no closed term* $\mathtt{u}$ *s.t. both* $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ *and* $\mathtt{uIdId}$ *reduce to pairs.*

Proof. By contradiction. Indeed, assume that there exist a closed term $\mathtt{u}$ such that both $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ and $\mathtt{uIdId}$ reduce to pairs. Since pairs are always typable, then $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ and $\mathtt{uIdId}$ are typable by Lem. 9.2. In any of the typing derivations of such terms, $\mathtt{u}$ occurs in a typed position, so that $\mathtt{u}$ turns out to be also typable.
Now, since $\mathtt{u}$ is typable and closed, then it reduces to a (typable and closed) canonical form $\mathtt{v} \in \mathcal{J}$ by Thm. 10. But $\mathtt{v}$ cannot be in $\mathcal{K}$, which only contains open terms. Moreover, $\mathtt{v}$ cannot be a pair, otherwise $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle \to^* \mathtt{v}\langle\mathtt{Id},\mathtt{Id}\rangle \to^* \langle\mathtt{v}_1,\mathtt{v}_2\rangle\langle\mathtt{Id},\mathtt{Id}\rangle \to^* \mathtt{fail}$ which contradicts (by Lem. 1) the fact that $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to a pair. We then have two possible forms for $\mathtt{v}$.
If $\mathtt{v} = \mathtt{s}[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{k}]$, where $\mathtt{s} \in \mathcal{J}$ and $\mathtt{k} \in \mathcal{K}$. Then $\mathtt{k}$ is an open term which implies $\mathtt{v}$ is an open term. Contradiction.
If $\mathtt{v} = \lambda\mathtt{p.s}$, where $\mathtt{s} \in \mathcal{J}$, then $\mathtt{p}$ is necessarily a variable, say $\mathtt{z}$, since otherwise $\mathtt{vId}$ reduces to $\mathtt{fail}$, and hence $\mathtt{uIdId} \to^* \mathtt{vIdId} \to^* \mathtt{fail}$, which contradicts (by Lem. 1) the fact that $\mathtt{uIdId}$ reduces to a pair. We analyze the possible forms of $\mathtt{s}$.

- If $\mathtt{s}$ is a pair, then $\mathtt{uIdId} \to^* (\lambda\mathtt{z.s})\mathtt{IdId} \to^* \mathtt{fail}$, which contradicts (by Lem. 1) the fact that $\mathtt{uIdId}$ reduces to a pair.
- If $\mathtt{s}$ is an abstraction, then $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle \to^* (\lambda\mathtt{z.s})\langle\mathtt{Id},\mathtt{Id}\rangle$ which reduces to an abstraction, contradicting (by Lem. 1) the fact that $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to a pair.
- If $\mathtt{s}$ is in $\mathcal{K}$, then $\mathtt{s} = \mathtt{xt}_1 \ldots \mathtt{t}_n$ with $n \geq 0$. Remark that $\mathtt{z} \neq \mathtt{x}$ is not possible since $\mathtt{v} = \lambda\mathtt{z.s}$ is closed. Then $\mathtt{z} = \mathtt{x}$. If $\mathtt{s} = \mathtt{z}$, then $\mathtt{uIdId}$ reduces to $\mathtt{Id}$ which contradicts (by Lem. 1) the fact that $\mathtt{uIdId}$ reduces to a pair. Otherwise, $\mathtt{s} = \mathtt{zt}_1 \ldots \mathtt{t}_n$ with $n \geq 1$, and thus $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to $\langle\mathtt{Id},\mathtt{Id}\rangle\mathtt{t}_1 \ldots \mathtt{t}_n \to^* \mathtt{fail}$, which contradicts again (by Lem. 1) the fact that $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to a pair.
- If $\mathtt{s}$ is $\mathtt{s}'[\langle\mathtt{p}_1,\mathtt{p}_2\rangle/\mathtt{k}]$, with $\mathtt{k} \in \mathcal{K}$, then $\mathtt{k} = \mathtt{zt}_1 \ldots \mathtt{t}_n$ with $n \geq 0$, since any other head variable for $\mathtt{k}$ would contradict $\mathtt{v}$ closed. Now, in the first case we have $\mathtt{uIdId}$ reduces to $\mathtt{fail}$ which contradicts (by Lem. 1) the fact that $\mathtt{uIdId}$ reduces to a pair. Otherwise, $\mathtt{k} = \mathtt{zt}_1 \ldots \mathtt{t}_n$ with $n \geq 1$ implies $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to $\mathtt{fail}$ which contradicts (by Lem. 1) the fact that $\mathtt{u}\langle\mathtt{Id},\mathtt{Id}\rangle$ reduces to a pair. ◀

The first non-observable term $\mathtt{t}_1$ in Ex. 12 could be ruled out by introducing a notion of *compatibility* between types and requiring multiset types to be composed only by compatible strict types. Unfortunately, we claim that a compatibility relation defined *syntactically*, let us call it $\mathtt{comp}$, cannot lead to a sound and complete characterization of observability. By "defined syntactically" we mean that the value of $\mathtt{comp}(\sigma \to \sigma', \rho \to \rho')$ should only depend on the values of $\mathtt{comp}(\sigma, \rho)$ and $\mathtt{comp}(\sigma', \rho')$. Another basic requirement of $\mathtt{comp}$ would be that every product type is incompatible with any functional type. The second non-observable term $\mathtt{t}_2$ in Ex. 12 is appropriate to illustrate our claim, by keeping in mind that any pair of types assignable to $\mathtt{x}$ in any typing derivation for $\mathtt{t}_2$ need to be incompatible.

Indeed, the shortest typing for $\mathtt{t}_2$ above is obtained by assigning to $\mathtt{x}$ the two types $[\,] \to o$ and $[\,] \to [\,] \to o$, and in order to state the incompatibility between them it would be necessary to define that $\mathtt{comp}(\sigma, \rho)$ and $\neg\mathtt{comp}(\sigma', \rho')$ imply $\neg\mathtt{comp}([\sigma] \to \sigma', [\rho] \to \rho')$. Another typing for $\mathtt{t}_2$ is obtained by assigning to $\mathtt{x}$ the two types $[o] \to o$ and $[\tau] \to [\tau] \to o$ respectively, where $\tau = [o] \to o$, so that $\neg\mathtt{comp}(\sigma, \rho)$ and $\neg\mathtt{comp}(\sigma', \rho')$ should imply $\neg\mathtt{comp}([\sigma] \to \sigma', [\rho] \to \rho')$. We conclude that $\neg\mathtt{comp}(\sigma', \rho')$ alone should imply $\neg\mathtt{comp}([\sigma] \to \sigma', [\rho] \to \rho')$. However, arrow types $[\sigma] \to \sigma'$ and $[\rho] \to \rho'$ having incompatible right-hand sides may very well be compatible. For instance, letting $\sigma = \sigma' = o$ and $\rho = \rho' = [o] \to o$, one gets two types for $\mathtt{Id}$

which need of course to be compatible. Hence, a *syntactic* characterization of such a notion of compatibility seems out of reach.

Fortunately, there exists a sound and complete *semantical* notion of compatibility between types, obtained *a posteriori* as follows: given two strict types π_1 and π_2, build the corresponding sets of inhabitants $\mathtt{T}(\emptyset, \pi_1)$ and $\mathtt{T}(\emptyset, \pi_2)$, using the inhabitation algorithm presented in Sec. 5. Then π_1 and π_2 are *semantically* compatible if and only if $\mathtt{T}(\emptyset, \pi_1) \cap \mathtt{T}(\emptyset, \pi_2)$ is non-empty.

While the inhabitation problem for (idempotent) intersection types is undecidable [17], it becomes decidable for non-idempotent intersection types [5], which is just a subsystem of our typing system $\mathcal{P}$ introduced in Sec. 3. We will prove in the following that inhabitation is also decidable for the non-trivial extension $\mathcal{P}$. We will then use this result for characterizing observability in the pattern calculus without referring to a complete syntactic characterization, which is not possible in this framework, as illustrated by Example 12.

5 Inhabitation for System $\mathcal{P}$

We now show a sound and complete algorithm to solve the inhabitation problem for System $\mathcal{P}$. Given a strict type σ, the inhabitation problem consists in finding a closed term $\mathtt{t}$ such that $\vdash \mathtt{t} : \sigma$ is derivable. We extend the problem to multiset types by defining $\mathtt{A}$ to be inhabited if and only if there is a closed term $\mathtt{t}$ such that $\vdash \mathtt{t} : \sigma_i$ for every $\sigma_i \in \mathtt{A}$. These notions will naturally be generalized later to non-closed terms.

We already noticed that the system $\mathcal{P}$ allows to type terms containing untyped subterms through the rule $(\to \mathtt{e})$ with $I = \emptyset$ and the rule $(\mathtt{sub})$ with $I = J = \emptyset$. In order to identify inhabitants in such cases we introduce a term constant Ω to denote a generic untyped subterm. Our inhabitation algorithm produces **approximate normal forms** $(\mathtt{a}, \mathtt{b}, \mathtt{c})$, also written *anf*, defined as follows:

$$\mathtt{a}, \mathtt{b}, \mathtt{c} ::= \Omega \mid \mathcal{N} \qquad \begin{array}{lcl} \mathcal{N} & ::= & \lambda \mathtt{p}.\mathcal{N} \mid \langle \mathtt{a}, \mathtt{b} \rangle \mid \mathcal{L} \mid \mathcal{N}[\langle \mathtt{p}, \mathtt{q} \rangle / \mathcal{L}] \\ \mathcal{L} & ::= & \mathtt{x} \mid \mathcal{L}\mathtt{a} \end{array}$$

Note that *anfs* do not contain redexes, differently from canonical forms. In particular, thanks to the reduction rule (r_4) (resp. (r_5)), they do not contain *head* (resp. *nested*) explicit matchings. This makes the inhabitation algorithm much more intuitive and simpler.

▶ **Example 14.** the term $\lambda \langle \mathtt{x}, \mathtt{y} \rangle.(\mathtt{x}(\mathtt{IdId}))[\langle \mathtt{z}_1, \mathtt{z}_2 \rangle / \mathtt{yId}]$ is canonical but not an *anf*, while $\lambda \langle \mathtt{x}, \mathtt{y} \rangle.(\mathtt{x}\Omega)[\langle \mathtt{z}_1, \mathtt{z}_2 \rangle / \mathtt{yId}]$ is an *anf*.

Anfs are ordered by the smallest contextual order $\leq$ such that $\Omega \leq \mathtt{a}$, for any $\mathtt{a}$. We also write $\mathtt{a} \leq \mathtt{t}$ when the term $\mathtt{t}$ is obtained from $\mathtt{a}$ by replacing each occurrence of Ω by a term of $\Lambda_{\mathtt{p}}$: For example $\mathtt{x}\Omega\Omega \leq \mathtt{x}(\mathtt{Id}\Delta)(\Delta\Delta)$ is obtained by replacing the first (resp. second) occurrence of Ω by $\mathtt{Id}\Delta$ (resp. $\Delta\Delta$).

Let $\mathcal{A}(\mathtt{t}) = \{\mathtt{a} \mid \exists \mathtt{u}\ \mathtt{t} \to^* \mathtt{u} \text{ and } \mathtt{a} \leq \mathtt{u}\}$ be the set of **approximants** of the term $\mathtt{t}$, and let $\bigvee$ denote the least upper bound with respect to $\leq$. We write $\uparrow_{i \in I} \mathtt{a}_i$ to denote the fact that $\bigvee \{\mathtt{a}_i\}_{i \in I}$ does exist. It is easy to check that, for every $\mathtt{t}$ and $\mathtt{a}_1, \ldots \mathtt{a}_n \in \mathcal{A}(\mathtt{t})$, $\uparrow_{i \in \{1,\ldots,n\}} \mathtt{a}_i$. An *anf* $\mathtt{a}$ is a **head subterm** of $\mathtt{b}$ if either $\mathtt{b} = \mathtt{a}$ or $\mathtt{b} = \mathtt{cc}'$ and $\mathtt{a}$ is a head subterm of $\mathtt{c}$. System $\mathcal{P}$ can also be trivially extended to give types to *anfs*, simply assuming that no type can be assigned to the constant Ω. It is easy to check that, if $\Gamma \vdash \mathtt{a} : \sigma$ and $\mathtt{a} \leq \mathtt{b}$ (resp. $\mathtt{a} \leq \mathtt{t}$) then $\Gamma \vdash \mathtt{b} : \sigma$ (resp. $\Gamma \vdash \mathtt{t} : \sigma$).

Given $\Pi \triangleright \Gamma \vdash \mathtt{t} : \tau$, where $\mathtt{t}$ is in Π-nf (*cf.* Sec. 3), $\mathcal{A}(\Pi)$ is the minimal approximant $\mathtt{b}$ of $\mathtt{t}$ such that $\Pi \triangleright \Gamma \vdash \mathtt{b} : \tau$. Formally, given $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$, where $\mathtt{t}$ is in Π-nf, the **minimal approximant of** Π, written $\mathcal{A}(\Pi)$, is defined by induction on $\mathtt{meas}(\Pi)$ as follows:

- $\mathcal{A}(\Gamma \vdash \mathtt{x} : \rho) = \mathtt{x}$; $\mathcal{A}(\Gamma \vdash \langle \mathtt{t}, \mathtt{u} \rangle : o) = \langle \Omega, \Omega \rangle$.
- If $\Pi \triangleright \Gamma \vdash \lambda \mathtt{p}.\mathtt{t} : \mathtt{A} \to \rho$ follows from $\Pi' \triangleright \Gamma' \vdash \mathtt{t} : \rho$, then $\mathcal{A}(\Pi) = \lambda \mathtt{p}.\mathcal{A}(\Pi')$, $\mathtt{t}$ being in Π'-nf.
- If $\Pi \triangleright \Gamma \vdash \langle \mathtt{t}, \mathtt{u} \rangle : \times_1(\tau)$ follows from $\Pi' \triangleright \Gamma \vdash \mathtt{t} : \tau$, then $\mathcal{A}(\Pi) = \langle \mathcal{A}(\Pi'), \Omega \rangle$, $\mathtt{t}$ being in Π'-nf. Similarly for a pair of type $\times_2(\tau)$.
- If $\Pi \triangleright \Gamma = \Gamma' +_{i \in I} \Delta_i \vdash \mathtt{tu} : \rho$ follows from $\Pi' \triangleright \Gamma' \vdash \mathtt{t} : [\sigma_i]_{i \in I} \to \rho$ and $(\Pi'_i \triangleright \Delta_i \vdash \mathtt{u} : \sigma_i)_{i \in I}$, then $\mathcal{A}(\Pi) = \mathcal{A}(\Pi')(\bigvee_{i \in I} \mathcal{A}(\Pi'_i))$
- If $\Pi \triangleright \Gamma = \Gamma' +_{i \in I} \Delta_i \vdash \mathtt{t}[\mathtt{p}/\mathtt{u}] : \tau$ follows from $\Pi' \triangleright \Gamma'' \vdash \mathtt{t} : \tau$ and $(\Psi_i \triangleright \Delta_i \vdash \mathtt{u} : \rho_i)_{i \in I}$, then $\mathcal{A}(\Pi) = \mathcal{A}(\Pi')[\mathtt{p}/\bigvee_{i \in I} \mathcal{A}(\Psi_i)]$

Remark that, in the application case of the definition above, the *anf* corresponding to $I = \emptyset$ is $\mathcal{A}(\Pi')\Omega$. Moreover, in the last case, $\mathtt{p}$ cannot be a variable, $\mathtt{t}$ being in Π-nf. A simple inspection of the typing rules for $\Vdash$ shows that in this case $I \neq \emptyset$.

▶ **Example 15.** Consider the following derivation Π:

$$\dfrac{\dfrac{\dfrac{\mathtt{y} : [[\,] \to o] \vdash \mathtt{y} : [\,] \to o}{\mathtt{y} : [[\,] \to o] \vdash \mathtt{y}(\Delta\Delta) : o} \quad \Vdash \langle \mathtt{z}_1, \mathtt{z}_2 \rangle : o \quad \dfrac{\mathtt{x} : [[\,] \to o] \vdash \mathtt{x} : [\,] \to o}{\mathtt{x} : [[\,] \to o] \vdash \mathtt{xId} : o}}{\mathtt{x} : [[\,] \to o]; \mathtt{y} : [[\,] \to o] \vdash \mathtt{y}(\Delta\Delta)[\langle \mathtt{z}_1, \mathtt{z}_2 \rangle / \mathtt{xId}] : o}}{\vdash \lambda \mathtt{xy}.\mathtt{y}(\Delta\Delta)[\langle \mathtt{z}_1, \mathtt{z}_2 \rangle / \mathtt{xId}] : [[\,] \to o] \to [[\,] \to o] \to o}$$

The minimal approximant of Π is $\lambda \mathtt{xy}.\mathtt{y}\Omega[\langle \mathtt{z}_1, \mathtt{z}_2 \rangle / \mathtt{x}\Omega]$.

A simple induction on $\mathtt{meas}(\Pi)$ allows to show the following:

▶ **Lemma 16.** *If* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$ *and* $\mathtt{t}$ *is in* Π*-nf, then* $\Pi \triangleright \Gamma \vdash \mathcal{A}(\Pi) : \sigma$.

5.1 The inhabitation algorithm

The inhabitation algorithm is presented in Fig. 2. As usual, in order to solve the problem for closed terms, it is necessary to extend the algorithm to open ones, so, given an environment Γ and a strict type σ, the algorithm builds the set $\mathtt{T}(\Gamma, \sigma)$ containing *all* the *anfs* $\mathtt{a}$ such that there exists a derivation $\Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$, with $\mathtt{a} = \mathcal{A}(\Pi)$, then stops[2]. Thus, our algorithm is not an extension of the classical inhabitation algorithm for simple types [4, 9]. In particular, when restricted to simple types, it constructs all the *anfs* inhabiting a given type, while the original algorithm reconstructs just the *long η-normal forms.* The algorithm uses four auxiliary predicates, namely

- $\mathtt{P}_{\mathcal{V}}(\mathtt{A})$, where $\mathcal{V}$ is a finite set of variables, contains the pairs $(\Gamma, \mathtt{p})$ such that (i) $\Gamma \Vdash \mathtt{p} : \mathtt{A}$, and (ii) $\mathtt{p}$ does not contain any variable in $\mathcal{V}$.
- $\mathtt{TI}(\Gamma, [\sigma_i]_{i \in I})$, contains all the *anfs* $\mathtt{a} = \bigvee_{i \in I} \mathtt{a}_i$ such that $\Gamma = +_{i \in I} \Gamma_i$, $\mathtt{a}_i \in \mathtt{T}(\Gamma_i, \sigma_i)$ for all $i \in I$, and $\uparrow_{i \in I} \mathtt{a}_i$.
- $\mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma, \sigma) \triangleright \tau$ contains all the *anfs* $\mathtt{a}$ such that $\mathtt{b}$ is a head subterm of $\mathtt{a}$, and such that if $\mathtt{b} \in \mathtt{T}(\Delta, \sigma)$ then $\mathtt{a} \in \mathtt{T}(\Gamma + \Delta, \tau)$.
- $\mathtt{HI}^{\Delta}_{\mathtt{b}}(\Gamma, [\sigma_i]_{i \in I}) \triangleright [\rho_i]_{i \in I}$ contains all the *anf* $\mathtt{a} = \bigvee_{i \in I} \mathtt{a}_i$ such that $\Delta = +_{i \in I} \Delta_i$, $\Gamma = +_{i \in I} \Gamma_i$, $\mathtt{a}_i \in \mathtt{H}^{\Delta}_{\mathtt{b}}(\Gamma, \sigma_i) \triangleright \rho_i$ and $\uparrow_{i \in I} \mathtt{a}_i$.

[2] It is worth noticing that, given Γ and σ, the set of *anfs* $\mathtt{a}$ such that there exists a derivation $\Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ is possibly infinite. However, the subset of those verifying $\mathtt{a} = \mathcal{A}(\Pi)$ is finite; they are the minimal ones, those generated by the inhabitation algorithm (this is proved in Lem. 19).

Note that the algorithm has different kinds of non-deterministic behaviours, *i.e.* different choices of rules can produce different results. Indeed, given an input (Γ, σ), the algorithm may apply a rule like $(\mathtt{Abs})$ in order to decrease the type σ, or a rule like $(\mathtt{Head})$ in order to decrease the environment Γ. Moreover, every rule (R) which is based on some decomposition of the environment and/or the type, like $(\mathtt{Subs})$, admits different applications. In what follows we illustrate the non-deterministic behaviour of the algorithm. For that, we represent a **run of the algorithm** as a tree whose nodes are labeled with the name of the rule applied.

▶ **Example 17.** We consider different inputs of the form $(\emptyset, \sigma)$, for different strict types σ. For every such input, we give an output and the corresponding run.

1. $\sigma = [[\alpha] \to \alpha] \to [\alpha] \to \alpha$.
   a. output: $\lambda \mathtt{xy}.\mathtt{xy}$, run: $\mathtt{Abs}(\mathtt{Abs}(\mathtt{Head}(\mathtt{Prefix}(\mathtt{TUn}(\mathtt{Head}(\mathtt{Final})), \mathtt{Final})), \mathtt{Varp}), \mathtt{Varp})$.
   b. output: $\lambda \mathtt{x}.\mathtt{x}$, run: $\mathtt{Abs}(\mathtt{Head}(\mathtt{Final}), \mathtt{Varp})$.
2. $\sigma = [[\,] \to \alpha] \to \alpha$. output: $\lambda \mathtt{x}.\mathtt{x}\Omega$, run: $\mathtt{Abs}(\mathtt{Head}(\mathtt{Prefix}(\mathtt{TUn}, \mathtt{Final})), \mathtt{Varp})$.
3. $\sigma = [[o] \to o, o] \to o$.
   a. output: $\lambda \mathtt{x}.\mathtt{xx}$, run: $\mathtt{Abs}(\mathtt{Head}(\mathtt{Prefix}(\mathtt{TUn}(\mathtt{Head}(\mathtt{Final})), \mathtt{Final})), \mathtt{Varp})$.
   b. Explicit substitutions may be used to consume some, or all, the resources in $[[o] \to o, o]$ output: $\lambda \mathtt{x}.\mathtt{x}[\langle \mathtt{y}, \mathtt{z}\rangle / \mathtt{x}\langle \Omega, \Omega\rangle]$, run:
   $\mathtt{Abs}(\mathtt{Subs}(\mathtt{HUn}(\mathtt{Prefix}(\mathtt{TUn}(\mathtt{Pair}), \mathtt{Final})), \mathtt{Pairp}(\mathtt{Weakp}, \mathtt{Weakp}), \mathtt{Head}(\mathtt{Final})), \mathtt{Varp})$.
   c. There are four additional runs, producing the following outputs:
   $\lambda \mathtt{x}.\mathtt{x}\langle \Omega, \Omega\rangle[\langle \mathtt{y}, \mathtt{z}\rangle / \mathtt{x}]$,
   $\lambda \mathtt{x}.\langle \Omega, \Omega\rangle[\langle \mathtt{y}, \mathtt{z}\rangle / \mathtt{xx}]$,
   $\lambda \mathtt{x}.\langle \Omega, \Omega\rangle[\langle \mathtt{y}, \mathtt{z}\rangle / \mathtt{x}][\langle \mathtt{w}, \mathtt{s}\rangle / \mathtt{x}\langle \Omega, \Omega\rangle]$,
   $\lambda \mathtt{x}.\langle \Omega, \Omega\rangle[\langle \mathtt{y}, \mathtt{z}\rangle / \mathtt{x}\langle \Omega, \Omega\rangle][\langle \mathtt{w}, \mathtt{s}\rangle / \mathtt{x}]$.

Along the recursive calls of the inhabitation algorithm, the parameters (type and/or environment) decrease strictly, for a suitable notion of measure, so that every run is finite:

▶ **Lemma 18.** *The inhabitation algorithm terminates.*

5.2 Soundness and completeness

We now prove soundness and completeness of our inhabitation algorithm.

▶ **Lemma 19.** $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma) \Leftrightarrow \exists \Pi \triangleright \Gamma \vdash \mathtt{a} : \sigma$ *such that* $\mathtt{a} = \mathcal{A}(\Pi)$.

Proof. The "only if" part is proved by induction on the rules in Fig. 2, and the "if" part is proved by induction on the definition of $\mathcal{A}(\Pi)$. In both parts, additional statements concerning the predicates of the inhabitation algorithm other than $\mathtt{T}$ are required, in order to strenghten the inductive hypothesis. ◀

▶ **Theorem 20** (Soundness and Completeness)**.**
1. *If* $\mathtt{a} \in \mathtt{T}(\Gamma, \sigma)$ *then, for all* $\mathtt{t}$ *such that* $\mathtt{a} \leq \mathtt{t}$, $\Gamma \vdash \mathtt{t} : \sigma$.
2. *If* $\Pi \triangleright \Gamma \vdash \mathtt{t} : \sigma$ *then there exists* $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ *such that* $\mathtt{t}'$ *is in* Π'*-nf, and* $\mathcal{A}(\Pi') \in \mathtt{T}(\Gamma, \sigma)$.

Proof. Soundness follows from Lem. 19 ($\Rightarrow$) and the fact that $\Gamma \vdash \mathtt{a} : \sigma$ and $\mathtt{a} \leq \mathtt{t}$ imply $\Gamma \vdash \mathtt{t} : \sigma$. For completeness we first apply Lem. 9.1 that guarantees the existence of $\Pi' \triangleright \Gamma \vdash \mathtt{t}' : \sigma$ such that $\mathtt{t}'$ is in Π'-nf, and then Lem. 16 and Lem. 19 ($\Leftarrow$). ◀

$$\frac{\mathtt{x} \notin \mathcal{V}}{(\emptyset; \mathtt{x}) \in_0 \mathtt{P}_{\mathcal{V}}([\,])}\ (\mathtt{Weakp}) \qquad \frac{\mathtt{x} \notin \mathcal{V}}{(\mathtt{x}:\mathtt{B}; \mathtt{x}) \in_0 \mathtt{P}_{\mathcal{V}}(\mathtt{B})}\ (\mathtt{Varp})$$

$$\frac{(\Gamma; \mathtt{p}) \in_i \mathtt{P}_{\mathcal{V}}(\mathtt{A}_1) \qquad (\Delta; \mathtt{q}) \in_j \mathtt{P}_{\mathcal{V}}(\mathtt{A}_2) \qquad \mathtt{p}\#\mathtt{q}}{(\Gamma; \Delta; \langle \mathtt{p}, \mathtt{q} \rangle) \in_1 \mathtt{P}_{\mathcal{V}}(\mathtt{A}_1 \mathbb{X} \mathtt{A}_2)}\ (\mathtt{Pairp})$$

$$\frac{\mathtt{a} \in \mathtt{T}(\Gamma; \Delta, \tau) \qquad \mathtt{A} = \mathtt{A}_1 \sqcup \mathtt{A}_2 \qquad (\Delta; \mathtt{p}) \in_i \mathtt{P}_{\mathtt{dom}(\Gamma)}(\mathtt{A}_1) \qquad \mathtt{supp}(\mathtt{A}_2) \subseteq \mathcal{S}(\mathtt{p})}{\lambda \mathtt{p}.\mathtt{a} \in \mathtt{T}(\Gamma, \mathtt{A} \to \tau)}\ (\mathtt{Abs})$$

$$\frac{(\mathtt{a}_i \in \mathtt{T}(\Gamma_i, \sigma_i))_{i \in I} \qquad \uparrow_{i \in I} \mathtt{a}_i}{\bigvee_{i \in I} \mathtt{a}_i \in \mathtt{TI}(+_{i \in I} \Gamma_i, [\sigma_i]_{i \in I})}\ (\mathtt{TUn}) \qquad \frac{(\mathtt{a}_i \in \mathtt{H}_{\mathtt{b}}^{\Delta_i}(\Gamma_i, \sigma_i) \rhd \rho_i)_{i \in I} \qquad \uparrow_{i \in I} \mathtt{a}_i}{\bigvee_{i \in I} \mathtt{a}_i \in \mathtt{HI}_{\mathtt{b}}^{+_{i \in I} \Delta_i}(+_{i \in I} \Gamma_i, [\sigma_i]_{i \in I}) \rhd [\rho_i]_{i \in I}}\ (\mathtt{HUn})$$

$$\frac{}{\langle \Omega, \Omega \rangle \in \mathtt{T}(\emptyset, o)}\ (\mathtt{Pair}) \quad \frac{\mathtt{a} \in \mathtt{T}(\Gamma, \tau)}{\langle \mathtt{a}, \Omega \rangle \in \mathtt{T}(\Gamma, \times_1(\tau))}\ (\mathtt{Prod1}) \quad \frac{\mathtt{a} \in \mathtt{T}(\Gamma, \tau)}{\langle \Omega, \mathtt{a} \rangle \in \mathtt{T}(\Gamma, \times_2(\tau))}\ (\mathtt{Prod2})$$

$$\frac{\mathtt{a} \in \mathtt{H}_{\mathtt{x}}^{\mathtt{x}:[\sigma]}(\Gamma, \sigma) \rhd \tau}{\mathtt{a} \in \mathtt{T}(\Gamma + (\mathtt{x}:[\sigma]), \tau)}\ (\mathtt{Head}) \quad \frac{\sigma = \tau}{\mathtt{a} \in \mathtt{H}_{\mathtt{a}}^{\Delta}(\emptyset, \sigma) \rhd \tau}\ (\mathtt{Final})$$

$$\frac{\Gamma = \Gamma_0 + \Gamma_1 \qquad \mathtt{b} \in \mathtt{TI}(\Gamma_0, \mathtt{A}) \qquad \mathtt{a} \in \mathtt{H}_{\mathtt{cb}}^{\Delta + \Gamma_0}(\Gamma_1, \sigma) \rhd \tau}{\mathtt{a} \in \mathtt{H}_{\mathtt{c}}^{\Delta}(\Gamma, \mathtt{A} \to \sigma) \rhd \tau}\ (\mathtt{Prefix})$$

$$\frac{\begin{array}{c}\Gamma = \Gamma_0 + \Gamma_1 \qquad \mathtt{c} \in \mathtt{HI}_{\mathtt{x}}^{\mathtt{x}:\mathtt{B}}(\Gamma_0, \mathtt{B}) \rhd \mathtt{F}(\mathtt{B}) \qquad \mathtt{F}(\mathtt{B}) = \mathtt{A}_1 \sqcup \mathtt{A}_2 (*) \\ (\Delta, \mathtt{p}) \in_1 \mathtt{P}_{\mathtt{dom}(\Gamma_0 + \Gamma_1 + (\mathtt{x}:\mathtt{B}))}(\mathtt{A}_1) \qquad \mathtt{supp}(\mathtt{A}_2) \subseteq \mathcal{S}(\mathtt{p}) \qquad \mathtt{b} \in \mathtt{T}(\Gamma_1; \Delta, \tau)\end{array}}{\mathtt{b}[\mathtt{p}/\mathtt{c}] \in \mathtt{T}(\Gamma + (\mathtt{x}:\mathtt{B}), \tau)}\ (\mathtt{Subs})$$

(*) where the operator $\mathtt{F}()$ is defined as follows:

$$\mathtt{F}(\alpha) \ := \ \alpha \qquad \mathtt{F}(\mathtt{A} \to \tau) \ := \ \mathtt{F}(\tau) \qquad \mathtt{F}([\,]) \ := \ [\,] \qquad \mathtt{F}([\sigma_i]_{i \in I}) \ := \ [\mathtt{F}(\sigma_i)]_{i \in I}$$

Figure 2 The inhabitation algorithm.

6 Characterizing Observability

We are now able to state the main result of this paper, *i.e.* the characterization of observability for the pattern calculus. The following lemma assures that types reflect correctly the structure of the data types.

▶ **Lemma 21.** *Let* t *be a closed and typable term, then*

- *If* t *has functional type, then* t *reduces to an abstraction.*
- *If* t *has product type, then* t *reduces to a pair.*

Proof. Let t be a closed and typable term. By Thm. 10 we know that t reduces to a (closed) canonical form in $\mathcal{J}$. The proof is by induction on the maximal length of such reduction sequences.

If t is already a canonical form, we analyze all the cases.

- If t is a variable, then this gives a contradiction with t closed.
- If t is a function, then the property trivially holds.

- If $\mathtt{t}$ is a pair, then the property trivially holds.
- If $\mathtt{t}$ is an application, then $\mathtt{t}$ has the form $\mathtt{x}\mathtt{t}_1 \dots \mathtt{t}_n$. Therefore at least $\mathtt{x}$ belongs to the set of free variables of $\mathtt{t}$, which leads to a contradiction with $\mathtt{t}$ closed.
- If $\mathtt{t}$ is a closure, *i.e.* $\mathtt{t} = \mathtt{u}[\langle \mathtt{p}_1, \mathtt{p}_2\rangle/\mathtt{v}]$, where $\mathtt{v} \in \mathcal{K}$ has the form $\mathtt{x}\mathtt{t}_1 \dots \mathtt{t}_n$, then at least $\mathtt{x}$ belongs to the set of free variables of $\mathtt{t}$, which leads to a contradiction with $\mathtt{t}$ closed.

Otherwise, $\mathtt{t} \to \mathtt{t}' \to^* \mathtt{u}$, where $\mathtt{u}$ is in $\mathcal{J}$. The term $\mathtt{t}'$ is also closed and typable (Lem. 9.1), then the *i.h.* gives the desired result for $\mathtt{t}'$, so the property holds also for $\mathtt{t}$. ◀

▶ **Theorem 22** (Characterizing Observability). *A term* $\mathtt{t}$ *is observable iff* $\Pi \triangleright \mathtt{x}_1 : \mathtt{A}_1; \dots ; \mathtt{x}_n : \mathtt{A}_n \vdash \mathtt{t} : \mathtt{B}_1 \to \dots \to \mathtt{B}_m \to \alpha$*, where* $n \geq 0, m \geq 0$*,* α *is a product type and all* $\mathtt{A}_1, \dots \mathtt{A}_n, \mathtt{B}_1, \dots \mathtt{B}_m$ *are inhabited.*

Proof. The left-to-right implication: if $\mathtt{t}$ is observable, then there exists a head-context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}] \to^* \langle \mathtt{u}, \mathtt{v}\rangle$. Since $\vdash \langle \mathtt{u}, \mathtt{v}\rangle : o$, we get $\Pi' \triangleright \vdash \mathtt{C}[\mathtt{t}] : o$ by Lem. 9.2. By definition $\mathtt{C}[\mathtt{t}] = (\lambda \mathtt{p}_1 ... \lambda \mathtt{p}_n.\mathtt{t})\mathtt{u}_1 ... \mathtt{u}_m$, so Π has a subderivation $\Pi' \triangleright \vdash \lambda \mathtt{p}_1 ... \lambda \mathtt{p}_n.\mathtt{t} : \mathtt{B}_1 \to \dots \to \mathtt{B}_m \to o$ (by rule $(\to \mathtt{e})$), where $\mathtt{B}_i$ is inhabited by $\mathtt{u}_i$ $(1 \leq i \leq m)$. Since $n \leq m$, Π' has a subderivation $\Pi'' \triangleright \Gamma \vdash \mathtt{t} : \mathtt{B}_{n+1} \to \dots \to \mathtt{B}_m \to o$ (by rule $(\to \mathtt{i})$), where $\Gamma|_{\mathtt{p}_i} \Vdash \mathtt{p}_i : \mathtt{B}_i$ $(1 \leq i \leq n)$. The result follows since $\mathtt{x}_1 : A_1, \dots, .\mathtt{x}_l : A_l \Vdash \mathtt{p} : B$ and B is inhabited implies that all the A_i are inhabited. The right-to-left implication: if $\mathtt{A}_1, \dots \mathtt{A}_n, \mathtt{B}_1, \dots \mathtt{B}_m$ are all inhabited, then there exist $\mathtt{u}_1, \dots \mathtt{u}_n, \mathtt{v}_1, \dots \mathtt{v}_m$ such that $\vdash \mathtt{u}_i : \sigma_i^j$ for every type σ_i^j of $\mathtt{A}_i$ $(1 \leq i \leq n)$ and $\vdash \mathtt{v}_i : \rho_i^j$ for every type ρ_i^j of $\mathtt{B}_i$ $(1 \leq i \leq m)$. Let $\mathtt{C} = (\lambda \mathtt{x}_1 \dots \mathtt{x}_n.\Box)\mathtt{u}_1 \dots \mathtt{u}_n \mathtt{v}_1 \dots \mathtt{v}_n$ be a head-context. We have $\vdash \mathtt{C}[\mathtt{t}] : \alpha$, which in turn implies that $\mathtt{C}[\mathtt{t}]$ reduces to a pair, by Lem. 21. Then the term $\mathtt{t}$ is observable by definition. ◀

The notion of observability is conservative with respect to that of solvability in λ-calculus.

▶ **Theorem 23** (Conservativity). *A* λ*-term* $\mathtt{t}$ *is solvable in the* λ*-calculus if and only if* $\mathtt{t}$ *is observable in* $\Lambda_{\mathtt{p}}$*.*

Proof.
- (if) Take an unsolvable λ-term $\mathtt{t}$ so that $\mathtt{t}$ does not have head normal-form. Then $\mathtt{t}$ (seen as a term of our calculus) has no canonical form, and thus $\mathtt{t}$ is not typable by Thm. 10. It turns out that $\mathtt{t}$ is not observable in $\Lambda_{\mathtt{p}}$ by Thm. 22.
- (only if) Take a solvable λ-term $\mathtt{t}$ so that there exist a head-context $\mathtt{C}$ such that $\mathtt{C}[\mathtt{t}]$ reduces to $\mathtt{Id}$, then it is easy to construct a head context $\mathtt{C}'$ such that $\mathtt{C}'[\mathtt{t}]$ reduces to a pair (just take $\mathtt{C}' = \mathtt{C}\ \langle \mathtt{t}_1, \mathtt{t}_2\rangle$ for some terms $\mathtt{t}_1, \mathtt{t}_2$). ◀

7 Conclusion and Further Work

We propose a notion of observability for pair pattern calculi which is conservative with respect to the notion of solvability for λ-calculus.

We provide a logical characterization of observable terms by means of typability *and* inhabitation.

Further work will be developed in different directions. As we already discussed in Sec. 2, different definitions of observability would be possible. We explored the one based on a lazy semantics, but it would be also interesting to obtain a full characterization based on a strict semantics. Another point to be developed is the definition of a suitable notion of head reduction, which, despite its relative simplicity, turn out to be quite cumbersome. On the semantical side, it is well known that non-idempotent intersection types can be used to supply a logical description of the relational semantics of λ-calculus [8, 14]. We would like to start from our type assignment system for building a denotational model of the pattern calculus. Last but not least, a challenging question is related to the characterization of observability in a more general framework of pattern λ-calculi allowing the patterns to be dynamic [10].

References

1 Thibaut Balabonski. On the implementation of dynamic patterns. In Eduardo Bonelli, editor, *HOR*, volume 49 of *EPTCS*, pages 16–30, 2010.

2 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, revised edition, 1984.

3 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

4 C. Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics*. PhD thesis, University of Wales Swansea, 1979.

5 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, *TCS*, LNCS. Springer, 2014. To appear.

6 Serenella Cerrito and Delia Kesner. Pattern matching as cut elimination. *Theoretical Computer Science*, 323(1-3):71–127, 2004.

7 Horatiu Cirstea, Germain Faure, and Claude Kirchner. A rho-calculus of explicit constraint application. *Higher-Order and Symbolic Computation*, 20(1-2):37–72, 2007.

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

9 J. Roger Hindley. *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Amsterdam, 2008.

10 Barry Jay and Delia Kesner. First-class patterns. *Journal of Functional Programming*, 19(2):191–225, 2009.

11 Wolfram Kahl. Basic pattern matching calculi: A fresh view on matching failure. In Yukiyoshi Kameyama and Peter Stuckey, editors, *FLOPS*, volume 2998 of *LNCS*, pages 276–290. Springer, 2004.

12 Jan-Willem Klop, Vincent van Oostrom, and Roel de Vrijer. Lambda calculus with patterns. *Theoretical Computer Science*, 398(1-3):16–31, 2008.

13 Jean Louis Krivine. *Lambda-Calculus, Types and Models*. Masson, Paris, and Ellis Horwood, Hemel Hempstead, 1993.

14 Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. Logical relational lambda-models. To appear in Mathematical Structures in Computer Science.

15 Barbara Petit. A polymorphic type system for the lambda-calculus with constructors. In Pierre-Louis Curien, editor, *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings*, volume 5608 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2009.

16 Simon Peyton-Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, Inc., 1987.

17 Pawel Urzyczyn. The emptiness problem for intersection types. *Journal of Symbolic Logic*, 64(3):1195–1215, 1999.

18 Vincent van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.