Pure Patterns Type Systems

Gilles Barthe and Horatiu Cirstea and Claude Kirchner and Luigi Liquori
LORIA, INRIA & University Nancy II: 54506 Vandoeuvre-lès-Nancy BP 239 Cedex France
INRIA: 06902 Sophia Antipolis BP 93 Cedex France
[Gilles.Barthe,Horatiu.Cirstea,Claude.Kirchner,Luigi.Liquori]@inria.fr

ABSTRACT

We introduce a new framework of algebraic pure type systems in which we consider rewrite rules as lambda terms with patterns and rewrite rule application as abstraction application with built-in matching facilities. This framework, that we call *"Pure Pattern Type Systems"*, is particularly well-suited for the foundations of programming (meta)languages and proof assistants since it provides in a fully unified setting higher-order capabilities and pattern matching ability together with powerful type systems. We prove some standard properties like confluence and subject reduction for the case of a syntactic theory and under a syntactical restriction over the shape of patterns. We also conjecture the strong normalization of typable terms. This work should be seen as a contribution to a formal connection between logics and rewriting, and a step towards new proof engines based on the Curry-Howard isomorphism.

Categories and Subject Descriptors

D.3.1 [**Formal Definitions and Theory**]: [Syntax, Semantics]; D.3.2 [**Language Classifications**]: [Applicative (functional) languages, Constraint and logic languages]; F.4.1 [**Mathematical Logic**]: [Lambda calculus and related systems, Logic and constraint programming, Mechanical theorem proving]

General Terms

Languages, Theory.

Keywords

Lambda-calculus, Patterns, Matching, Rewriting, Pure Type Systems, Curry-Howard, Logics.

1. INTRODUCTION

λ-calculus and term rewriting provide two fundamental computational paradigms that had a deep influence on the development of programming and specification languages, and on proof environments. Starting from Klop's groundbreaking work on higher-order rewriting [19], and because of their complementarity, many frameworks have been designed with a view to integrate these two formalisms.

This integration has been handled either by enriching first-order rewriting with higher-order capabilities or by adding to λ-calculus algebraic features. In the first case, we find the works on CRS [20] and other higher-order rewriting systems [30, 23], in the second case the works on combination of λ-calculus with term rewriting [2, 7, 12, 17] to mention only a few.

In some works, the λ-calculus has been extended with *pattern abstractions* that generalize λ-abstractions by binding structured expressions instead of variables. Such pattern abstractions are commonly used to compile case-expressions in functional programming languages [24] and to provide term calculi for sequent calculi [18]. For example, the pattern abstractions $\lambda 0.0$ and $\lambda \mathsf{succ}(N).N$ are used to compile the predecessor function.

In other works, like those on higher-order rewriting systems, such pattern abstractions are extended to *generalized abstractions* by binding arbitrary expressions instead of patterns. Apart their expressive power, generalized abstractions correspond to a form of higher-order natural deduction, where (parts of) proof trees are discharged instead of assumptions.

Although such extended abstractions are a firmly grounded artifact both in logic and in programming language design and implementation, they lack established foundations. As a first step towards these foundations, we provide a framework for studying (type systems for) extended abstractions. Concretely, we start from a term calculus with generalized abstractions, which provides a very rich computational model that goes far beyond λ-calculus, and endorse it with a parameterized type system as in Pure Type Systems.

Pure Type Systems (PTS) [4, 15] provide a concise and unifying view of many typed λ-calculi and logics that occur in the literature, and offer a generic framework for the study of typed λ-calculi à la Church. Pure Type Systems have a well-understood theory and enjoy good meta-theoretical properties, and hence offer an appealing setting in which to specify the kernel of functional programming languages, see *e.g.* [1, 6, 25, 28], and proof assistants based on the Curry-Howard isomorphism, see *e.g.* [4, 5].

In *Pure Pattern Type Systems* ($\mathsf{P}^2\mathsf{TS}$), the framework presented here, the usual λ-abstraction of Pure Type Systems

POPL'03, January 15–17, 2003, New Orleans, Louisiana, USA.

$\lambda X{:}A.B$ of type $\Pi X{:}A.C$ is replaced by a generalized abstraction $\lambda A{:}\Delta.B$ of type $\Pi A{:}\Delta.C$, where A is an arbitrary term (in jargon a *pattern*) and Δ is a context in which are declared the bound variables of $\lambda A{:}\Delta.B$.

Adding patterns to PTSs may seem innocuous. However, defining a conservative extension of PTSs that provides support for patterns and enjoys basic meta-theoretical properties, like confluence and subject reduction, turns out to be a laborious task. Indeed, both properties fail, even in a simply-typed setting, when no restrictions are imposed on the term formation nor on the reduction strategy, see Examples 4 and 5.

For confluence, the problem was already remarked in [29], where confluence is recovered by imposing a so-called "Rigid Pattern Condition" on patterns. Such a condition is directly applicable to $\mathsf{P^2TS}$s, but requires patterns to be linear. Linearity of patterns is an overly strong assumption in an explicitly typed language with polymorphism and dependent types. Indeed, many polymorphic functions, *e.g.* the function $\lambda cons(T\ X\ nil(T)){:}\Delta.X$ that returns the only element of a one-element list, involve non-linear patterns. Hence van Oostrom's condition must be relaxed so that non-linear patterns are allowed. This is the first major hurdle with extending PTSs to $\mathsf{P^2TS}$s.

Another problem is caused by the necessity to allow for free variables in the pattern of an abstraction. For example, $\lambda cons(T\ X\ nil(T)){:}\Delta.X$ should be of type $list(T)\triangleright T$, and should have T as free variable. In order for reduction to be closed under substitution, and for typing to be closed under substitution and reduction, we must therefore allow for patterns that are not in normal form and allow for reduction in patterns. Furthermore, matching should be modified so that the free variables of a pattern whose types are not defined in the context are considered as free in the corresponding abstractions and handled as constants by the matching algorithm. This is the second major hurdle with extending PTSs to $\mathsf{P^2TS}$s.

Therefore, the main contributions of this paper are:

- to provide adequate notions of patterns, substitutions and matching that overcome the afore-mentioned hurdles. One technical tool used for this purpose is the notion of delayed matching constraint, see Section 2, and the possibility for patterns in abstractions to evolve (by reduction or substitution) during execution;
- on the basis of these definitions, to propose a well-behaved extension of PTSs that supports abstraction over patterns, and that enjoys many fundamental properties of PTSs, including confluence, subject reduction, conservativity over PTSs and consistency for normalizing $\mathsf{P^2TS}$s.

The remaining of the paper is organized as follows: Section 2 presents the syntax of $\mathsf{P^2TS}$ and related notions such as free variables and substitutions. Matching systems are also introduced together with a (syntactic) matching algorithm. The dynamic semantics of $\mathsf{P^2TS}$ and some examples of reductions are given in Section 3. The typing rules are presented in Section 4 and exemplified by two type derivations. The meta-theory including some properties of the matching algorithm, the conditions ensuring the confluence and the preservation of types, as well as some other classical type-related properties are stated and discussed in Section 5. Finally, we conclude by conjecturing some properties of (possible extensions of) $\mathsf{P^2TS}$ and give some perspectives to this work.

2. SYNTAX

The syntax of the $\mathsf{P^2TS}$ extends that of PTSs with structures and abstractions on patterns [10]. The contexts defining the types of the free variables of these patterns are given explicitly as part of the abstraction.

2.1 Notations and Conventions

In this paper, we consider the meta-symbols λ (function abstraction), and Π (product, or type abstraction), and "," (structure operator), and the (hidden) "$\bullet$" (application operator). We assume that the application operator "$\bullet$" associates to the left, while the other operators associate to the right. The priority of "$\bullet$" is higher than that of "$[\ll].$" which is higher than that of "Π "and "λ" which are, in turn, of higher priority than the ",".

The symbol $\checkmark$ranges over the set $\{\lambda\ \Pi\}$, while the symbol $\fatsemi$ range over the set $\{\bullet\ ,\}$. The symbols $A, B, C, \ldots$ range over the set $\mathcal{T}$ of terms, the symbols $X, Y, Z, \ldots$ range over an infinite set $\mathcal{V}$ of variables, the symbols $a, b, c, f, \ldots$ range over a set $\mathcal{K}$ of constants, the symbol s ranges over a set $\mathcal{S}$ of sorts. The symbols $\alpha, \beta, \ldots$ range over $\mathcal{K}$ and $\mathcal{V}$. The symbols Γ, Δ, Σ range over the set $\mathcal{C}$ of contexts. The symbols $\sigma, \tau, \theta, \ldots$ range over substitutions. All symbols can be indexed.

As usual, we work modulo the "α*-conversion*" and adopt Barendregt's "*hygiene-convention*" [3], *i.e.* free and bound variables have different names.

2.2 Pseudo-Terms

The definition of pseudo-terms and pseudo-contexts is parametrized by the set $\mathcal{K}$ of constants, and $\mathcal{S}$ of sorts ($\mathcal{S} \subseteq \mathcal{K}$), and $\mathcal{V}$ of variables, and $\mathcal{P}$ of patterns ($\mathcal{V} \subseteq \mathcal{P} \subseteq \mathcal{T}$).

$$\mathcal{C} \ ::= \ \emptyset \mid \mathcal{C}, \mathcal{V}{:}\mathcal{T} \mid \mathcal{C}, \mathcal{K}{:}\mathcal{T}$$

$$\mathcal{T} \ ::= \ \mathcal{V} \mid \mathcal{K} \mid \lambda\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid \Pi\mathcal{P}{:}\mathcal{C}.\mathcal{T} \mid [\mathcal{P} \ll_{\mathcal{C}} \mathcal{T}].\mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid \mathcal{T}, \mathcal{T}$$

A term of the form $\checkmark A{:}\Delta.B$ is an *abstraction* (resp. *product abstraction*) with pattern A, body B and context Δ. The term $[A \ll_{\Delta} C].B$ is a *delayed matching constraint* with pattern A, body B and argument C. A term of the form (A, B) is called a *structure* with elements A and B.

In an *application* $(A\,B)$, the term A represents the function, while the term B represents the argument. The application of a constant function, say f, to a term A will be usually denoted by $f(A)$, following the algebraic "folklore"; this convention can be "curryfied" in order to denote a function taking multiple arguments, *e.g.* $f(A_1 \cdots A_n) \triangleq f\ A_1\ \cdots A_n$. If we set $\mathcal{P} \equiv \mathcal{V}$, then we recover a syntax that is equivalent to PTS, see Theorem 5.

In $\mathsf{P^2TS}$ we need to recast the definition of free variables.

Definition 1 (Free Variables). *The set* Fv *of free variables is inductively defined on terms (and pointwise extended to contexts) as follows:*

$$\mathsf{Fv}(a) \triangleq \emptyset$$

$$\mathsf{Fv}(X) \triangleq \{X\}$$

$$\mathsf{Fv}(\checkmark A{:}\Delta.B) \triangleq (\mathsf{Fv}(A) \cup \mathsf{Fv}(B) \cup \mathsf{Fv}(\Delta)) \setminus \mathsf{Dom}(\Delta)$$

$$\begin{array}{lcl}
\mathsf{Fv}([A \ll_\Delta C].B) & \triangleq & \mathsf{Fv}((\lambda A{:}\Delta.B)\, C) \\
\mathsf{Fv}(A \fatsemi B) & \triangleq & \mathsf{Fv}(A) \cup \mathsf{Fv}(B) \\
\mathsf{Fv}(\emptyset) & \triangleq & \emptyset \\
\mathsf{Fv}(\Delta, \alpha{:}A) & \triangleq & \mathsf{Fv}(\Delta) \cup \mathsf{Fv}(A)
\end{array}$$

A term A is closed *if* $\mathsf{Fv}(A) = \emptyset$.

Intuitively the context Δ in $\surd A{:}\Delta.B$ (resp. $[A \ll_\Delta C].B$) contains the type declarations of some (but not all) of the free variables (not necessarily the type of constants) appearing in the pattern A, *i.e.* $\mathsf{Dom}(\Delta) \subseteq \mathsf{Fv}(A)$. These variables are bound in the (pattern and body of the) abstraction and the reduction of an abstraction application strongly depends on them, all the other variables being handled as constants. The free variables of A not declared in Δ are not bound in B but could be bound outside the scope of the abstraction itself. As a simple example, in the abstraction $\lambda f(X\ Y){:}(X{:}i).g(X\ Y)$, the X variable is bound, while the Y variable is free.

As in ordinary systems dealing with dependent types, we suppose that in $\Gamma, \alpha{:}A$, the α does not appear free in Γ, and A. The set Bv of bound variables of a term is the complementary of the set of free variables *w.r.t.* the set of variables of the respective term. Since we work modulo α-conversion, we suppose that all the bound variables of a term have different names and therefore, the domains of all contexts are distinct. The following example presents some legal pseudo-terms and their free variables according to Definition 1.

Example 1 (Some pseudo-terms).

- *(Cons) Let* $A \triangleq \lambda cons(X nil){:}\Delta.X$, *with* $\Delta \triangleq X{:}i$. *Then* $\mathsf{Fv}(A) = \emptyset$.
- *(PolyCons_0) Let* $A \triangleq \lambda cons(T\, X\, nil(T)){:}\Delta.X$, *with* $\Delta \triangleq \emptyset$. *Then* $\mathsf{Fv}(A) = \{T, X\}$.
- *(PolyCons_1) Let* $A \triangleq \lambda cons(T\, X\, nil(T)){:}\Delta.X$, *with* $\Delta \triangleq X{:}T$. *Then* $\mathsf{Fv}(A) = \{T\}$.
- *(PolyCons_2) Let* $A \triangleq \lambda cons(T\, X\, nil(T)){:}\Delta.X$, *with* $\Delta \triangleq T{:}*, X{:}T$. *Then* $\mathsf{Fv}(A) = \emptyset$.
- *(λ-Pattern) Let* $A \triangleq \lambda(\lambda X{:}\Gamma.X\,Y){:}\Delta.Z$, *with* $\Gamma \triangleq X{:}\Pi Z{:}i.i$, *and* $\Delta \triangleq Y{:}i$. *Then* $\mathsf{Fv}(A) \triangleq \{Z\}$.

2.3 Substitutions

We adapt the classical notion of simultaneous substitution application to deal with the new forms of abstraction introduced in the $\mathsf{P^2TS}$.

Definition 2 (Substitutions). *A substitution σ is of a finite map $\{A_1/X_1 \dots A_m/X_m\}$. The application $B\sigma$ of a substitution σ to a term B is defined as follows:*

$$\begin{array}{lcl}
a\sigma & \triangleq & a \\
X_i\sigma & \triangleq & \begin{cases} A_i & \text{if } X_i \in \mathsf{Dom}(\sigma) \\ X_i & \text{otherwise} \end{cases} \\
(\surd A{:}\Delta.B)\sigma & \triangleq & \surd A\sigma{:}\Delta\sigma.B\sigma \\
([A \ll_\Delta B].C)\sigma & \triangleq & [A\sigma \ll_{\Delta\sigma} B\sigma].C\sigma \\
(A \fatsemi B)\sigma & \triangleq & A\sigma \fatsemi B\sigma \\
\emptyset\sigma & \triangleq & \emptyset \\
(\Delta, \alpha{:}A)\sigma & \triangleq & \Delta\sigma, \alpha{:}A\sigma
\end{array}$$

We let

$$\mathsf{Dom}(\sigma) = \{X_1, \dots, X_m\} \qquad \mathsf{CoDom}(\Delta) = \bigcup_{i=1\dots m} \mathsf{Fv}(A_i)$$

A substitution σ is independent from Δ, written $\sigma \not\propto \Delta$, if $\mathsf{Dom}(\sigma) \cap \mathsf{Dom}(\Delta) = \emptyset$ *and* $\mathsf{CoDom}(\sigma) \cap \mathsf{Dom}(\Delta) = \emptyset$. *By abuse of language this notation can be used with a list of variables instead of a context.*

2.4 Matching

$\mathsf{P^2TSs}$ feature pattern abstractions whose application requires solving matching problems. For the purpose of this paper, we focus on syntactic matching.

Definition 3 (Matching Systems).

1. *A matching system* $\mathsf{T} \triangleq \bigwedge_{i=0\dots n} A_i \prec\!\!\prec^{\Sigma}_{\Delta_i} B_i$ *is a conjunction of match equations, where $\wedge$ is idempotent, associative and commutative.*

2. *A matching system* T *is solved by the substitution σ if* $\mathsf{Dom}(\sigma) \subseteq \mathsf{Dom}(\Sigma) \setminus \mathsf{Dom}(\Delta)$ *and for all* $i = 0 \dots n$, $A_i\sigma \equiv B_i$.

3. *A matching system* T *is in* solved *form when it satisfies the following conditions:*

   (a) $\mathsf{T} \triangleq \bigwedge_{i=0\dots n} X_i \prec\!\!\prec^{\Sigma}_{\Delta_i} C_i \bigwedge_{j=0\dots m} a_j \prec\!\!\prec^{\Sigma}_{\Delta_j} a_j$;

   (b) for all $h, k = 0 \dots n$, $X_h \equiv X_k$ *implies* $C_h \equiv C_k$;

   (c) for all $i = 0 \dots n$, $X_i \in \mathsf{Dom}(\Delta_i)$ *or* $X_i \notin \mathsf{Dom}(\Sigma)$ *implies* $X_i \equiv C_i$;

   (d) for all $i = 0 \dots n$, $\mathsf{Fv}(C_i) \cap \mathsf{Dom}(\Delta_i) \neq \emptyset$ *implies* $X_i \equiv C_i$.

The domain of the context Σ specifies the variables concerned by the matching; all the other variables of A_i are treated as constants. The domain of the context Δ_i specifies the variables that should not be considered (*i.e.* handled as constants) when solving the equation $A_i \prec\!\!\prec^{\Sigma}_{\Delta_i} B_i$. In the matching systems that should be solved in $\mathsf{P^2TSs}$ the domains Σ and Δ are disjoint since they define the types of the bound variables of different terms.

Intuitively, the conditions can be read as follows:

(3a) imposes that a *solved* matching system should be a conjunction of equations of the shape "variable-term", or "constant-constant";

(3b) checks for clashes due to a variable matching different terms;

(3c) checks that bound variables are not instantiated (σ behaves as the identity on all variables occurring in $\mathsf{Dom}(\Delta_i)$) and that only the variables explicitly specified as "matchable" (*a.k.a.* declared in the context of the abstraction) lead to useful replacements;

(3d) checks that free variables are not instantiated by bound variables. This condition is essential to guarantee that substitution does not capture bound variables.

Our matching algorithm is adapted from classical higher-order matching (and unification) algorithms [13, 16, 21], in order to account for the specificities of $\mathsf{P^2TSs}$: syntactic matching that does not involve the reduction calculus, abstractions over patterns, and free variables in patterns.

$$(Lbd/Prod) \quad (\checkmark A_1{:}\Delta.B_1) \prec\!\!\prec_\Gamma^\Sigma (\checkmark A_2{:}\Delta.B_2) \rightsquigarrow A_1 \prec\!\!\prec_{\Gamma,\Delta}^\Sigma A_2 \wedge B_1 \prec\!\!\prec_{\Gamma,\Delta}^\Sigma B_2$$

$$(Delay) \quad [A_1 \ll_\Delta C_1].B_1 \prec\!\!\prec_\Gamma^\Sigma [A_2 \ll_\Delta C_2].B_2 \rightsquigarrow A_1 \prec\!\!\prec_{\Gamma,\Delta}^\Sigma A_2 \wedge B_1 \prec\!\!\prec_{\Gamma,\Delta}^\Sigma B_2 \wedge C_1 \prec\!\!\prec_\Gamma^\Sigma C_2$$

$$(Appl/Struct) \quad (A_1 \,\S\, B_1) \prec\!\!\prec_\Gamma^\Sigma (A_2 \,\S\, B_2) \rightsquigarrow A_1 \prec\!\!\prec_\Gamma^\Sigma A_2 \wedge B_1 \prec\!\!\prec_\Gamma^\Sigma B_2$$

Figure 1: Matching Algorithm $\mathcal{A}lg$

DEFINITION 4 (MATCHING ALGORITHM $\mathcal{A}lg$). *The relation $\rightsquigarrow$ is the compatible relation induced by the rules of Figure 1; the relation $\twoheadrightarrow$ is defined as the reflexive and transitive closure of $\rightsquigarrow$. If $\mathsf{T} \twoheadrightarrow \mathsf{T}'$, with T' a matching system in* solved *form then, we say that the matching algorithm $\mathcal{A}lg$ (taking as input the system T) succeeds.*

The matching algorithm is clearly terminating (since all rules decrease the size of terms) and deterministic (no critical pairs), and of course, it works modulo α-conversion and Barendregt's hygiene-convention.

Starting form a given solved matching system of the form $\mathsf{T} \triangleq \bigwedge_{i=0\dots n} X_i \prec\!\!\prec_{\Delta_i}^\Sigma A_i \bigwedge_{j=0\dots m} a_j \prec\!\!\prec_{\Delta_j}^\Sigma a_j$, the corresponding substitution $\{A_1/X_1 \cdots A_n/X_n\}$ is exhibited. By abuse of notation, if $\mathcal{A}lg$ succeeds on $A \prec\!\!\prec_\Delta^\Sigma B$ we write $\exists\sigma.\mathcal{A}lg(A \prec\!\!\prec_\Delta^\Sigma B)$ and the substitution $\sigma_{(A \prec\!\!\prec_\Delta^\Sigma B)}$ denotes the substitution corresponding to the solved matching system computed by $\mathcal{A}lg$ with input $A \prec\!\!\prec_\Delta^\Sigma B$. To ease the notation, Σ and Δ are omitted if they are not essential when verifying (the conditions on the) the solved form of a matching system like, in particular when Δ is empty or Σ contains all the free variables of the left-hand sides of the matching equations.

EXAMPLE 2 (RUNS OF $\mathcal{A}lg$). *Let $\Gamma = \{X{:}i\}$, and $\Delta = \{Y{:}i\}$, and $\Theta = \{Z{:}i\}$, and an unspecified context Σ.*

1. $f(X) \prec\!\!\prec_\emptyset^\Gamma f(3) \twoheadrightarrow f \prec\!\!\prec_\emptyset^\Gamma f \wedge X \prec\!\!\prec_\emptyset^\Gamma 3$
   (solved by $\sigma = \{3/X\}$);
2. $f(X) \prec\!\!\prec_\emptyset^\emptyset f(3) \twoheadrightarrow f \prec\!\!\prec_\emptyset^\emptyset f \wedge X \prec\!\!\prec_\emptyset^\emptyset 3$
   (not solvable since it does not satisfy condition 3c);
3. $\lambda X{:}\Gamma.X \prec\!\!\prec_\emptyset^\Sigma \lambda X{:}\Gamma.Y \rightsquigarrow X \prec\!\!\prec_\Gamma^\Sigma X \wedge X \prec\!\!\prec_\Gamma^\Sigma Y$
   (not solvable since it does not satisfy condition 3c);
4. $\lambda X{:}\Gamma.X \prec\!\!\prec_\emptyset^\Sigma \lambda 3{:}\Gamma.3 \rightsquigarrow X \prec\!\!\prec_\Gamma^\Sigma X \wedge X \prec\!\!\prec_\Gamma^\Sigma 3$
   (not solvable since it does not satisfy condition 3c);
5. $\lambda X{:}\Gamma.Y \prec\!\!\prec_\emptyset^\Sigma \lambda X{:}\Gamma.\lambda Z{:}\Theta.f(X\,Z) \rightsquigarrow$
   $X \prec\!\!\prec_\Gamma^\Sigma X \wedge Y \prec\!\!\prec_\Gamma^\Sigma \lambda Z{:}\Theta.f(X\,Z)$
   (not solvable since it does not satisfy condition 3d);
6. $\lambda X{:}\Gamma.f(X\,Y) \prec\!\!\prec_\emptyset^\Delta \lambda X{:}\Gamma.f(X\,3) \twoheadrightarrow$
   $X \prec\!\!\prec_\Gamma^\Delta X \wedge f \prec\!\!\prec_\Gamma^\Delta f \wedge X \prec\!\!\prec_\Gamma^\Delta X \wedge Y \prec\!\!\prec_\Gamma^\Delta 3$
   (solved by $\sigma = \{3/Y\}$; all equations satisfy 3c, 3d);
7. $[f(X) \ll_\Gamma f(Y)].X \prec\!\!\prec_\emptyset^\Delta [f(X) \ll_\Gamma f(3)].X \twoheadrightarrow$
   $f \prec\!\!\prec_\Gamma^\Delta f \wedge X \prec\!\!\prec_\Gamma^\Delta X \wedge X \prec\!\!\prec_\Gamma^\Delta X \wedge f \prec\!\!\prec_\emptyset^\Delta f \wedge Y \prec\!\!\prec_\emptyset^\Delta 3$
   (solved by $\sigma = \{3/Y\}$);

The matching algorithm $\mathcal{A}lg$ is sound and closed by substitution, see Theorem 1 and Theorem 2.

$$(\rho) \quad (\lambda A{:}\Delta.B)C \;\to_\rho\; [A \ll_\Delta C].B$$

$$(\sigma) \quad [A \ll_\Delta C].B \;\to_\sigma\; B\sigma_{(A \prec\!\!\prec_\emptyset^\Delta C)}$$

$$(\delta) \quad (A, B)\,C \;\to_\delta\; A\,C, B\,C$$

Figure 2: Top-level Rules of the $\mathsf{P^2TS}$

3. DYNAMIC SEMANTICS

3.1 Top-level Rules

The top-level rules are presented in Figure 2. The central idea of the (ρ) rule of the calculus is that the application of a term $\lambda A{:}\Delta.B$ to a term C, reduces to the delayed matching constraint $[A \ll_\Delta C].B$, while the application of the (σ) rule consists in solving the matching equation $A \prec\!\!\prec_\emptyset^\Delta C$, and applying the obtained substitution (if it exists) to the the term B. One should notice that the subsequent matching concerns only the variables bound by the corresponding context. If no solution exists, then the (σ) rule cannot be fired and therefore the term $[A \ll_\Delta C].B$ is not reduced. The rule (δ) deals with the distributivity of the application on the structures built with the "," constructor.

It is important to remark that if A is a variable, then the subsequent combination of (ρ) and (σ) rules corresponds exactly to the (β) rule of the λ-calculus, and variable manipulations in substitutions are handled externally, using α-conversion and Barendregt's hygiene convention if necessary.

3.2 (One/Many)-Step, Congruence

The next definition introduces the classical notions of one-step, many-steps, and congruence relation of $\to_{\rho\sigma\delta}$.

DEFINITION 5 (ONE/MANY-STEPS, CONGRUENCE). *Let $\mathsf{Ctx}[-]$ be any term $\mathcal{T}$ with a "single hole" inside, and let $\mathsf{Ctx}[A]$ be the result of filling the hole with the term A;*

1. *the one-step evaluation $\mapsto_{\rho\sigma\delta}$ is defined by the following inference rule, where $\to_{\rho\sigma\delta} \equiv \to_\rho \cup \to_\sigma \cup \to_\delta$:*

$$\frac{A \to_{\rho\sigma\delta} B}{\mathsf{Ctx}[A] \mapsto_{\rho\sigma\delta} \mathsf{Ctx}[B]} \quad (\mathsf{Ctx}[-])$$

2. *the many-step evaluation $\mapsto\!\!\!\!\to_{\rho\sigma\delta}$ and congruence relation $=_{\rho\sigma\delta}$ are respectively defined as the reflexive-transitive and reflexive-symmetric-transitive closure of $\mapsto_{\rho\sigma\delta}$.*

3.3 Two Simple (Untyped) Examples

We present the reduction of two terms using different evaluation strategies (outermost *vs.* innermost) and yielding in the first case a "successful" result (*i.e.* containing no delayed matching constraint) and in the second one an "unsuccessful" one. We underline redexes to be reduced.

EXAMPLE 3 (TWO SIMPLE EVALUATIONS IN $\mathsf{P^2TS}$).
A successful computation

- $\underline{(\lambda f(X).(\lambda 3.3)X)\, f(3)} \mapsto_\rho \underline{[f(X) \ll f(3)].((\lambda 3.3)\, X)} \mapsto_\sigma ((\lambda 3.3)\, X)\{3/X\} \triangleq \underline{(\lambda 3.3)\, 3} \mapsto_\rho \underline{[3 \ll 3].3} \mapsto_\sigma 3\{\ \} \triangleq 3$
- $(\lambda f(X).\underline{(\lambda 3.3)X})\, f(3) \mapsto_\rho \underline{(\lambda f(X).[3 \ll X].X)\, f(3)} \mapsto_\rho \underline{[f(X) \ll f(3)].([3 \ll X].X)} \mapsto_\sigma ([3 \ll X].X)\{3/X\} \triangleq \underline{[3 \ll 3].3} \mapsto_\sigma 3\{\ \} \triangleq 3$

An unsuccessful computation

- $\underline{(\lambda f(X).(\lambda 3.3)X)\, f(4)} \mapsto_\rho \underline{[f(X) \ll f(4)].((\lambda 3.3)\, X)} \mapsto_\sigma ((\lambda 3.3)\, X)\{4/X\} \triangleq \underline{(\lambda 3.3)\, 4} \mapsto_\rho [3 \ll 4].3$
- $(\lambda f(X).\underline{(\lambda 3.3)X})\, f(4) \mapsto_\rho \underline{(\lambda f(X).[3 \ll X].3)\, f(4)} \mapsto_\rho \underline{[f(X) \ll f(4)].([3 \ll X].3)} \mapsto_\sigma ([3 \ll X].3)\{4/X\} \triangleq [3 \ll 4].3$

It is worth noticing that the term $[3 \ll 4].3$ represents *de facto* a computation failure, which can be read as follows:

> *"The result would be* 3 *but at run-time the program tried to match* 3 *against* 4*, yielding the (dirty) result* $[3 \ll 4].3$*"*

The capability of $\mathsf{P^2TS}$ to record failures is directly inherited from previous versions of the rewriting-calculus, where a special symbol Null denoted computational failures (*a.k.a.* exceptions); different mechanisms dealing with exception handling, such as *e.g.* `try` $\mathcal{T}$ `catch` $[\mathcal{T} \ll \mathcal{T}]$ `with` $\mathcal{T}$, have been studied in [11, 14], in an untyped framework.

4. STATIC SEMANTICS

This section presents the type system underneath $\mathsf{P^2TS}$; as for PTSs, the formalism of $\mathsf{P^2TS}$ is parameterized by a notion of specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{S}$ is a subset of $\mathcal{C}$ and contains the *sorts*, $\mathcal{A} \subseteq \mathcal{S}^2$ is a set of *axioms*, and $\mathcal{R} \subseteq \mathcal{S}^3$ is a set of *rules*.

We require all specifications to be *functional* [4], *i.e.* for every $s_1, s_2, s_2', s_3, s_3' \in \mathcal{S}$, the following holds:

$$\begin{array}{lllll} (s_1, s_2) \in \mathcal{A} & \text{and} & (s_1, s_2') \in \mathcal{A} & \text{implies} & s_2 \equiv s_2' \\ (s_1, s_2, s_3) \in \mathcal{R} & \text{and} & (s_1, s_2, s_3') \in \mathcal{R} & \text{implies} & s_3 \equiv s_3'. \end{array}$$

Furthermore, we let $\mathcal{S}^\top$ denote the set of *topsorts*, i.e.

$$\mathcal{S}^\top = \{s \in \mathcal{S} \mid \forall s' \in \mathcal{S}.\ (s, s') \notin \mathcal{A}\}$$

and define a variant of delayed matching constraint as follows:

$$[A \ll_\Delta C]^\top.B = \begin{cases} B & \text{if } B \in \mathcal{S}^\top \\ [A \ll_\Delta C].B & \text{otherwise} \end{cases}$$

4.1 A "Guided Tour" of Type Rules

The notion of type derivation in $\mathsf{P^2TS}$ involves a judgment of the shape:

$$\Gamma \vdash A : B$$

The type system is defined by the rules of Figure 3. Some rules are inherited from PTSs but other deserve a brief explanation, notably on how they differ from their PTS counterpart:

- The $(Conv)$ rule is relaxed *w.r.t.* the usual one can find in usual PTS (where normally we have $\Gamma \vdash C : s$);

$$\frac{(s_1, s_2) \in \mathcal{A}}{\emptyset \vdash s_1 : s_2}\ (Axioms)$$

$$\frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : C}{\Gamma \vdash A, B : C}\ (Struct)$$

$$\frac{\Gamma \vdash A : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}A \vdash \alpha : A}\ (Start)$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha{:}C \vdash A : B}\ (Weak)$$

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : D \qquad B =_{\rho\delta} C}{\Gamma \vdash A : C}\ (Conv)$$

$$\frac{\Gamma, \Delta \vdash B : C \qquad \Gamma \vdash \Pi A{:}\Delta.C : s}{\Gamma \vdash \lambda A{:}\Delta.B : \Pi A{:}\Delta.C}\ (Abs)$$

$$\frac{\begin{array}{ll}\Gamma \vdash C : s_1 & (s_1, s_2, s_3) \in \mathcal{R} \\ \Gamma, \Delta \vdash A : C & \Gamma, \Delta \vdash B : s_2\end{array}}{\Gamma \vdash \Pi A{:}\Delta.B : s_3}\ (Prod)$$

$$\frac{\Gamma \vdash A : \Pi C{:}\Delta.D \qquad \Gamma, \Delta \vdash C : E \qquad \Gamma \vdash B : E}{\Gamma \vdash A\ B : [C \ll_\Delta B].D}\ (Appl)$$

$$\frac{\Gamma, \Delta \vdash A : E \qquad \Gamma, \Delta \vdash C : D \qquad \Gamma \vdash B : D}{\Gamma \vdash [C \ll_\Delta B].A : [C \ll_\Delta B]^\top.E}\ (Subst)$$

Figure 3: The Type Rules for $\mathsf{P^2TS}$

- The $(Struct)$ rule says that a structure A, B, where $A : C$ and $B : C$ can be typed with type C, hence forcing all subterms to be of the same type;
- The (Abs) rule, which deals λ-abstractions in which we bind over (non trivial) patterns, requires in particular that the pattern and body of the abstraction are typable in the extended context Γ, Δ;
- Likewise the $(Prod)$ rule, which deals with product types, requires that the pattern and codomain of the product is typable in the extended context Γ, Δ;
- The $(Appl)$ rule, which deals with applications, imposes that the resulting type in the conclusion features delayed matching. In case the delayed matching can be successfully solved by a run of the matching algorithm $\mathcal{A}lg(C \prec\!\!\ll^\Delta_\emptyset B)$, one can recover the expected type by applying the conversion rule;
- The $(Subst)$ rule deals with terms in which a *delayed matching constraint* occurs *hard-coded* into the term (and its type); this rule is crucial to ensure the Subject Reduction property for the top-level reduction rule (ρ). Observe that we require, as for the (Abs) rule, that the pattern and body of the delayed matching abstraction

$$
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma,\Delta \vdash (\lambda 3{:}\emptyset.3) : \Pi 3{:}\emptyset.i\end{array} \quad \boxed{5} \quad \boxed{6}}{\Gamma,\Delta \vdash (\lambda 3{:}\emptyset.3)\, X : [3 \ll_{\emptyset} X].i} \quad \dfrac{\begin{array}{c}\boxed{3} \quad (*,*,*) \in \mathcal{R}\\ \Gamma \vdash [Z \ll_{\Sigma} X].i : *\\ \Gamma,\Delta \vdash [3 \ll_{\emptyset} X].i : *\end{array}}{\Gamma \vdash \Pi f(X){:}\Delta.[3 \ll_{\emptyset} X].i : *}}{\Gamma \vdash \lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X : \Pi f(X){:}\Delta.[3 \ll_{\emptyset} X].i} \quad \boxed{3} \quad \boxed{4}}{\Gamma \vdash (\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : [f(X) \ll_{\Delta} f(3)].[3 \ll_{\emptyset} X].i} \quad \boxed{1} \quad \boxed{2}}{\Gamma \vdash (\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : i}
$$

where $\boxed{1} \triangleq [f(X) \ll_{\Delta} f(3)].[3 \ll_{\emptyset} X].i =_{\rho\sigma\delta} i$, and $\boxed{2} \triangleq \Gamma \vdash i : *$, and $\boxed{3} \triangleq \Gamma, \Delta \vdash f(X) : [Z \ll_{\Sigma} X].i$, and $\boxed{4} \triangleq \Gamma \vdash f(3) : [Z \ll_{\Sigma} 3].i$, and $\boxed{5} \triangleq \Gamma, \Delta \vdash X : i$, and $\boxed{6} \triangleq \Gamma, \Delta \vdash 3 : i$.

Figure 4: A Typing Derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma,\Sigma,\Delta \vdash i : *\end{array}}{\Gamma,\Sigma,\Delta \vdash X : i} \quad \dfrac{\boxed{1} \quad \boxed{2} \quad \boxed{3} \quad (*,*,*) \in \mathcal{R}}{\Gamma,\Sigma \vdash \Pi P{:}\Delta.T : *}}{\Gamma,\Sigma \vdash \lambda P{:}\Delta.X : \Pi P{:}\Delta.T} \quad \boxed{4}}{\Gamma \vdash (\lambda T{:}\Sigma.\lambda P{:}\Delta.X) : \Pi T{:}\Sigma.\Pi P{:}\Delta.T} \quad \boxed{5} \quad \boxed{6}}{\Gamma \vdash (\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i : [T \prec\!\!\ll_{\Sigma} i].\Pi P{:}\Delta.T} \quad \boxed{7} \quad \boxed{8}}{\Gamma \vdash (\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i : \Pi P_i{:}\Delta_i.i} \quad \begin{array}{c}\vdots\\ \Gamma,\Delta_i \vdash P_i : list(i)\end{array} \quad \begin{array}{c}\vdots\\ \Gamma \vdash P' : list(i)\end{array}}{\Gamma \vdash (\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i\, P' : [P_i \prec\!\!\ll_{\Delta_i} P'].i} \quad \boxed{9} \quad \boxed{10}}{\Gamma \vdash (\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i\, P' : i}
$$

where $\boxed{1}$ denotes $\Gamma, \Sigma, \Delta \vdash P : list(T)$, which can be derived as follows: (we omit some simple derivations and we disregard simple applications of $(Conv)$ rule after every application of an $(Appl)$ rule).

$$
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma,\Sigma,\Delta \vdash \Pi T{:}\Sigma.T \triangleright list(T) \triangleright list(T) : *\end{array}}{\Gamma,\Sigma,\Delta \vdash cons : \Pi T{:}\Sigma.T \triangleright list(T) \triangleright list(T)} \quad \boxed{3}}{\Gamma,\Sigma,\Delta \vdash cons(T) : T \triangleright list(T) \triangleright list(T)} \quad \dfrac{\boxed{3}}{\Gamma,\Sigma,\Delta \vdash X : T}}{\Gamma,\Sigma,\Delta \vdash cons(T)\, X : list(T) \triangleright list(T)} \quad \dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma,\Sigma,\Delta \vdash \Pi T{:}\Sigma.list(T) : *\end{array}}{\Gamma,\Sigma,\Delta \vdash nil : \Pi T{:}\Sigma.list(T)} \quad \boxed{3}}{\Gamma,\Sigma,\Delta \vdash nil(T) : list(T)}}{\Gamma,\Sigma,\Delta \vdash P : list(T)}
$$

and $\boxed{2} \triangleq \Gamma, \Sigma \vdash list(T) : *$, and $\boxed{3} \triangleq \Gamma, \Sigma, \Delta \vdash T : *$, and $\boxed{4} \triangleq \Gamma \vdash \Pi T{:}\Sigma.\Pi P{:}\Delta.T : *$, and $\boxed{5} \triangleq \Gamma, \Sigma \vdash T : *$, and $\boxed{6} \triangleq \Gamma \vdash i : *$, and $\boxed{7} \triangleq \Gamma \vdash \Pi P_i{:}\Delta_i.i : *$, and $\boxed{8} \triangleq [T \prec\!\!\ll_{\Sigma} i].\Pi P{:}\Delta.T =_{\rho\sigma\delta} \Pi P_i{:}\Delta_i.i$, $\boxed{9} \triangleq \Gamma \vdash i : *$, and $\boxed{10} \triangleq [P_i \prec\!\!\ll_{\Delta_i} P'].i =_{\rho\sigma\delta} i$.

Figure 5: Another Typing Derivation

are typable in the extended context Γ, Δ. Note how the variant of delayed matching is used in the type of the conclusion, as plain delayed matching would lead to a failure of Subject Reduction (indeed one would have $\vdash [X \ll_{\Delta} Y].\star : [X \ll_{\Delta} Y].\Box$ but not $\vdash \star : [X \ll_{\Delta} Y].\Box$).

4.2 Two Typing Derivations

To better help the reader in the comprehension of $\mathsf{P^2TS}$, this subsection shows *in extenso* two typing derivations: the first derivation (in Figure 4) allows us to derive the term

$$\Gamma \vdash (\lambda f(X){:}\Delta.(\lambda 3{:}\emptyset.3)X)\, f(3) : i$$

where $\Gamma \triangleq i{:}*, f{:}\Pi Z{:}\Sigma.i, 3{:}i, 4{:}i$, and $\Delta \triangleq X{:}i$, and $\Sigma \triangleq Z{:}i$.

The second derivation (in Figure 5) was directly inspired by one of the referees; it allows to type some basic polymorphic list operators, like `car/cdr/head` typable using higher-order constructor types, like $list(T)/cons(T)/null(T)$, *i.e.*

$$\Gamma \vdash (\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i\, P' : i$$

where we let $A \triangleright B \triangleq \Pi X{:}A.B$ when $X \notin \mathsf{Fv}(B)$, and

$$\begin{array}{lcl} \Delta & \triangleq & X{:}T \quad \text{and} \quad \Delta_i \triangleq X{:}i \quad \text{and} \quad \Sigma \triangleq T{:}* \\ \Gamma & \triangleq & i{:}*, 3{:}i, list{:}* \triangleright *, nil{:}\Pi T{:}\Sigma.list(T), \\ & & cons{:}\Pi T{:}\Sigma.T \triangleright list(T) \triangleright list(T) \\ P & \triangleq & cons(T)\, X\, nil(T) \quad \text{and} \quad P_i \triangleq cons(i)\, X\, nil(i) \\ P' & \triangleq & cons(i)\, 3\, nil(i) \end{array}$$

denotes an applications of the polymorphic function (`head`) which takes as argument first the type i to instantiate into a list of integers, and then the singleton integer list `'(3)` (denoted by the term P) and produce as result the integer value 3. Therefore, the polymorphic pattern P denotes the shape of a singleton polymorphic list. It easy to verify that $(\lambda T{:}\Sigma.(\lambda P{:}\Delta.X))\, i\, P' \mapsto\!\!\!\to_{\rho\delta} 3$, and that $\not\mapsto\!\!\!\to (\lambda P{:}\Delta.X))\, P'$ (*i.e.* the polymorphism need to be instantiated first).

5. METATHEORY

This section collects all the meta-theory of $\mathsf{P^2TS}$: more precisely, we prove Confluence, Subject Reduction, Conservativity over PTSs, and Consistency of normalizing $\mathsf{P^2TS}$s.

5.1 Properties of the Matching Algorithm

The matching algorithm $\mathcal{A}lg$ presented in Section 2.4 is sound, that is, the substitution corresponding to the matching system in solved form computed by $\mathcal{A}lg$ solves (according to Definition 3) the initial matching system.

THEOREM 1 (SOUNDNESS OF $\mathcal{A}lg$). *If* $\exists\sigma.\mathcal{A}lg(A \prec\!\!\prec^{\Sigma}_{\Delta} B)$, *then* $\mathsf{Dom}(\sigma) \subseteq \mathsf{Dom}(\Sigma) \setminus \mathsf{Dom}(\Delta)$ *and* $A\sigma \equiv B$.

PROOF. *The first property is obtained immediately by construction. For the equivalence, the proof is done by induction on the structure of the term* A. □

The classical syntactic matching algorithms are obviously closed by substitutions containing no variables involved in the matching. In $\mathsf{P^2TS}$ some of the variables of the pattern can be free and consequently, the corresponding matching should handle them as constants. Thus, we should ensure that the replacement of these variables by other terms does not affect the success of the matching algorithm. Nevertheless, the substitutions that we consider should not lead to "variable capture" and we impose the appropriate conditions that are satisfied when dealing with matching systems generated from well-formed $\mathsf{P^2TS}$ terms.

THEOREM 2 ($\mathcal{A}lg$ IS CLOSED BY SUBSTITUTION). *If* $\exists\sigma.\mathcal{A}lg(A \prec\!\!\prec^{\Sigma}_{\Gamma} B)$ *and the substitution* τ *is such that* $\tau \not\!\lhd \Sigma$, *and* $\tau \not\!\lhd \mathsf{Bv}(A)$, *and* $\tau \not\!\lhd \mathsf{Dom}(\Gamma)$ *then,* $\exists\theta.\mathcal{A}lg(A\tau \prec\!\!\prec^{\Sigma\tau}_{\Gamma\tau} B\tau)$ *and* $\sigma_{A\tau \prec\!\!\prec^{\Sigma\tau}_{\Gamma\tau} B\tau} = (\sigma_{A \prec\!\!\prec^{\Sigma}_{\Gamma} B})\tau$.

PROOF. *By induction on the structure of* A. □

5.2 Confluence - The Rigid Pattern Condition

When no restrictions are imposed either on the term formation or on the reduction strategy, confluence fails. Therefore, a suitable condition Cond on $\mathcal{P}$ should be used in order to ensure that $\mathsf{P^2TS}$ are well-behaved extensions of plain PTS.

The shape of patterns in $\mathsf{P^2TS}$ abstractions is essential in order to avoid bizarre non-confluent reductions and thus, in order to recover the good properties, Cond should ensure that these patterns satisfy certain properties. For the scope of this paper we use a condition inspired from the *Rigid Pattern Condition* (RPC) that was first formalized in [29].

The *Rigid Pattern Condition* is sufficient for obtaining the "diamond property" of the (parallel) reduction and hence the confluence of the reduction in $\mathsf{P^2TS}$. As explained in [29], this kind of rigid pattern condition does not characterize the shape of patterns, but syntactic characterizations of the terms satisfying this condition can be given.

We introduce the RPC which turns out to be sufficient to prove that the parallel reduction $\Rightarrow_{\rho\delta}$ (defined below) satisfies the diamond property. The condition we present differs from the original one of [29], since it has been customized to better fit for our $\mathsf{P^2TS}$ and, in particular, the possibility to reduce the patters that can possibly contain free variables.

Let us first define the notion of parallel reduction for our $\mathsf{P^2TS}$.

DEFINITION 6 (PARALLEL REDUCTION). *The parallel reduction, denoted by* $\Rightarrow_{\rho\delta}$, *is inductively defined as follows:*

$$\begin{array}{ll} (Par_1) & \alpha \Rightarrow_{\rho\delta} \alpha \\ (Par_2) & A_1 \,\S\, B_1 \Rightarrow_{\rho\delta} A_2 \,\S\, B_2 \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ \S \in \{\bullet\ ,\} \\ (Par_3) & \checkmark A_1{:}\Delta_1.B_1 \Rightarrow_{\rho\delta} \checkmark A_2{:}\Delta_2.B_2 \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ \Delta_1 \Rightarrow_{\rho\delta} \Delta_2,\ \checkmark \in \{\lambda\ \Pi\} \\ (Par_4) & (\lambda A_1{:}\Delta_1.B_1)C_1 \Rightarrow_{\rho\delta} [A_2 \ll_{\Delta_2} C_2].B_2 \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ C_1 \Rightarrow_{\rho\delta} C_2,\ \Delta_1 \Rightarrow_{\rho\delta} \Delta_2 \\ (Par_5) & (A_1, B_1)\, C_1 \Rightarrow_{\rho\delta} A_2\, C_2, B_2\, C_2 \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ C_1 \Rightarrow_{\rho\delta} C_2 \\ (Par_6) & [A_1 \ll_{\Delta_1} B_1].C_1 \Rightarrow_{\rho\delta} [A_2 \ll_{\Delta_2} B_2].C_2 \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ C_1 \Rightarrow_{\rho\delta} C_2,\ \Delta_1 \Rightarrow_{\rho\delta} \Delta_2 \\ (Par_7) & [A_1 \ll_{\Delta_1} B_1].C_1 \Rightarrow_{\rho\delta} C_2\sigma_{(A_2 \prec\!\!\prec^{\Delta_2}_{\emptyset} B_2)} \\ & \quad \textit{if } A_1 \Rightarrow_{\rho\delta} A_2,\ B_1 \Rightarrow_{\rho\delta} B_2,\ C_1 \Rightarrow_{\rho\delta} C_2,\ \Delta_1 \Rightarrow_{\rho\delta} \Delta_2, \\ & \quad\quad \exists\sigma_{(A_2 \prec\!\!\prec^{\Delta_2}_{\emptyset} B_2)} \end{array}$$

The rules $(Par_1), \ldots, (Par_3)$ indicate that the relation $\Rightarrow_{\rho\delta}$ includes the identity on $\lambda^{\ll}$-terms, *i.e.* $A \Rightarrow_{\rho\delta} A$ holds for all $A \in \mathcal{T}$. Rules $(Par_4), \ldots, (Par_7)$ deal with reductions. Intuitively, $A \Rightarrow_{\rho\delta} B$ means that B is obtained from A, by simultaneous contractions of some $(\rho\delta)$-redexes possibly overlapping one each other.

DEFINITION 7. *Define the mapping* $\diamond$ *by induction:*

$$\begin{array}{rcl} \alpha^{\diamond} & \triangleq & \alpha \\ (A, B)^{\diamond} & \triangleq & A^{\diamond}, B^{\diamond} \\ (\checkmark A{:}\Delta.B)^{\diamond} & \triangleq & \checkmark A^{\diamond}{:}\Delta^{\diamond}.B^{\diamond} \\ (AB)^{\diamond} & \triangleq & A^{\diamond} B^{\diamond} \\ & & \textit{if } A \textit{ is not abstraction or structure} \\ ((\lambda A{:}\Delta.B)C)^{\diamond} & \triangleq & [A^{\diamond} \ll_{\Delta^{\diamond}} C^{\diamond}].B^{\diamond} \\ ((A, B)C)^{\diamond} & \triangleq & (A^{\diamond} C^{\diamond}), (B^{\diamond} C^{\diamond}) \\ ([A \ll_{\Delta} B].C)^{\diamond} & \triangleq & [A^{\diamond} \ll_{\Delta^{\diamond}} B^{\diamond}].C^{\diamond} \\ & & \textit{if } \sigma_{(A^{\diamond} \ll_{\Delta^{\diamond}} B^{\diamond})} \textit{ does not exist} \\ ([A \ll_{\Delta} B].C)^{\diamond} & \triangleq & C^{\diamond}\sigma_{(A^{\diamond} \ll_{\Delta^{\diamond}} B^{\diamond})} \end{array}$$

The rules describing the parallel reduction and the mappings for contexts and substitutions are not presented since they are completely syntax dependent and naturally follow from the other rules. Using these two latter definitions we introduce the RPC and its extension ERPC.

Intuitively, the RPC ensures that the form of a pattern is preserved by reduction even when some of its variables are instantiated. This means that the only redexes introduced when instantiating the variables of a pattern by a term are the ones already present in the term.

Definition 8 (Rigid Pattern Condition (RPC)). *The term P satisfies* RPC *if for all $A, B, \sigma_1 \Rightarrow_{\rho\delta} \sigma_2$:*

1. *$P\sigma_1 \Rightarrow_{\rho\delta} B$ implies $B \equiv P\sigma_2$,*
2. *$A \Rightarrow_{\rho\delta} P\sigma_2$ implies $A^\diamond \equiv P\sigma_1^\diamond$.*

The RPC ensures not only that the form of patterns is preserved but also that if a term is reduced to the instantiation of a pattern then, its $\diamond$-mapping is also an instantiation of the same pattern.

The RPC can be seen as a coherence condition between the reduction and the matching, *i.e.* if a given matching system involving "rigid" patterns has a solution then the system obtained by reducing the terms of the initial one has a (corresponding) solution as well. Worthy to notice that $\mathcal{V}$ obviously satisfies RPC, but $\mathcal{T}$ do not.

Since in $\mathsf{P^2TS}$ the variables that are not bound in a pattern by the corresponding domain are treated as constants (in the matching algorithm), the conditions imposed by RPC should not be enforced on these variables. Therefore, we relax this condition and introduce the *Extended Rigid Pattern Condition* (ERPC) that is defined *w.r.t.* a given context.

Definition 9 (Extended RPC, ERPC(Δ)). *The term E satisfies* ERPC(Δ) *if there exist P satisfying* RPC *and $\sigma \not\lhd \Delta$ such that $E \equiv P\sigma$.*

The pattern A of $\mathsf{P^2TS}$ abstractions of the form $\surd A{:}\Delta.B$ and $\mathsf{P^2TS}$ delayed matching constraints of the form $[A \ll_\Delta C].B$ should satisfy ERPC(Δ). In the examples and properties presented in this section we sometimes consider that all variables of a pattern are bound in the corresponding abstractions since free variables are considered as constants in the matching algorithm.

At this point of the paper, the RPC condition appears somewhat obscure, since it involves the definition of the parallel reduction. Nevertheless, to help the reader, we can (syntactically) characterize an "honest" subset $\mathcal{P}$ of $\mathcal{T}$ which properly contains $\mathcal{V}$ and satisfies RPC.

Definition 10 (Characterization of $\mathcal{P}_{\mathsf{RPC}}$). *Let $\mathsf{Nf}(\rho\delta)$ be the set of terms that cannot be reduced by one of the rules of $\mathsf{P^2TS}$. Define*

$$\begin{aligned}\mathcal{P}_{\mathsf{RPC}} \triangleq \{A \in \mathsf{Nf}(\rho\delta) \mid\ & A \textit{ contains no subterms of the form}\\ & (X\ B) \textit{ s.t. } X \in \mathsf{Fv}(A) \textit{ and no subterms of the form}\\ & [B \ll_\Delta C].D, \textit{ with } \mathsf{Fv}(A) \cap (\mathsf{Fv}(B) \cup \mathsf{Fv}(C)) \neq \emptyset\}\end{aligned}$$

This characterization imposes that a pattern from the set $\mathcal{P}$ contains no so-called "active" variables that are free and that all patterns and arguments of its delayed matching constraints are closed.

Starting from this characterization for RPC, we can also characterize a set of terms

$$\mathcal{P} \triangleq \{P\sigma \mid P \in \mathcal{P}_{\mathsf{RPC}} \text{ and } \sigma \not\lhd \Delta\}$$

satisfying ERPC(Δ).

One can notice that this latter characterization is closed by substitution (with a (co)domain containing no variables from the domain of Δ) and thus, is preserved by reduction in the patterns of $\mathsf{P^2TS}$ abstractions.

The following examples show that when considering patterns that do not satisfy the above characterization, the confluence is lost.

Example 4. *(All type annotations are omitted).*

1. *Patterns containing free "active" variables like X in the term $(\lambda(XY).X)((\lambda Z.Z)a)$ lead to a non-confluent reductions:*

$$\begin{tikzcd}[column sep=small] & (\lambda(XY).X)((\lambda Z.Z)a) \arrow[dl, Rightarrow, "\rho\delta"'] \arrow[dr, Rightarrow, "\rho\delta"] & \\ [XY \ll (\lambda Z.Z)a].X \arrow[d, Rightarrow, "\rho\delta"'] & & (\lambda(XY).X)([Z \ll a].Z) \arrow[d, Rightarrow, "\rho\delta"] \\ (\lambda Z.Z) & & (\lambda(XY).X)a \end{tikzcd}$$

2. *When a pattern containing free variables in the argument of a delayed matching constraint is used, non-confluent derivations can be obtained:*

$$\begin{tikzcd}[column sep=small] (\lambda([a \ll Y].b).Y)([a \ll a].b) \arrow[d, Rightarrow, "\rho\delta"'] \arrow[dr, Rightarrow, "\rho\delta"] & \\ [[a \ll Y].b \ll [a \ll a].b].Y \arrow[d, Rightarrow, "\rho\delta"'] & (\lambda([a \ll Y].b).Y)(b) \arrow[d, Rightarrow, "\rho\delta"] \\ a & [[a \ll Y].b \ll b].Y \end{tikzcd}$$

Worthy noticing that the original characterization of RPC of [29] forbids patterns that are "non-linear", *i.e.* with multiple occurrences of free variables inside, (*e.g.* in $f(X\,X)$). The above characterization does not enforce this condition and this is suitable especially when dealing with terms typable with polymorphic and dependent types.

Lemma 1. *If $P \in \mathcal{P}_{\mathsf{RPC}}$ then P satisfies* RPC.

Proof. *We show that if $P \in \mathcal{P}_{\mathsf{RPC}}$ then, for all $B, \sigma_1 \Rightarrow_{\rho\delta} \sigma_2$*

1. *$P\sigma_1 \Rightarrow_{\rho\delta} B$ implies $B \equiv P\sigma_2$,*
2. *$B \Rightarrow_{\rho\delta} P\sigma_2$ implies $B^\diamond \equiv P\sigma_1^\diamond$.*

For the first point we should notice that a pattern should be in normal form and thus, the only redexes are introduced by the substitution. On the other hand, redexes that do not exist in σ can be created only by instantiating free "active" variables by a structure or an abstraction or by instantiating the variables of a delayed matching constraint and making solvable a matching equation. None of these cases is possible since P contains no free "active" variables and no free *variables in the pattern and argument of a delayed matching constraint and thus, the property holds.*

For the second point, we proceed by induction on the definition of $B \Rightarrow_{\rho\sigma\delta} P\sigma_2$. The case of (Par_1) is trivial and all the other cases except for (Par_7) follow easily by IH. For (Par_7) we have $[A_1 \ll_{\Delta_1} B_1].C_1 \Rightarrow_{\rho\sigma\delta} C_2\sigma_{(A_2 \prec\!\!\prec_{\emptyset}^{\Delta_2} B_2)}$ with $A_1, B_1, C_1 \in \mathcal{P}_{\mathsf{RPC}}$. It can be shown easily that if $A_1 \Rightarrow_{\rho\sigma\delta} A_2$, and $B_1 \Rightarrow_{\rho\sigma\delta} B_2$, and $\Delta_1 \Rightarrow_{\rho\sigma\delta} \Delta_2$, and $\exists\sigma_2.\mathcal{A}lg(A_2 \prec\!\!\prec_{\emptyset}^{\Delta_2} B_2)$, then $\exists\sigma_1^\diamond.\mathcal{A}lg(A_1^\diamond \prec\!\!\prec_{\emptyset}^{\Delta_1^\diamond} B_1^\diamond)$ and $\sigma_2 \Rightarrow_{\rho\sigma\delta} \sigma_1^\diamond$. Then, the property follows by IH and Lemma 4. $\square$

In order to prove the confluence of the $\mapsto_{\rho\sigma\delta}$ relation it is enough to prove the diamond property of the $\mapsto\!\!\!\!\to_{\rho\sigma\delta}$ relation. We can prove this latter property directly or we can propose a relation whose transitive closure is the $\mapsto\!\!\!\!\to_{\rho\sigma\delta}$ relation and that satisfies the diamond property and obtain as an immediate consequence the diamond property for the $\mapsto\!\!\!\!\to_{\rho\sigma\delta}$ relation. In what follows we use the latter approach and chose as target relation the parallel reduction introduced in Definition 6.

Lemma 2 (Relations). $\mapsto_{\rho\sigma\delta} \subseteq \Rightarrow_{\rho\sigma\delta} \subseteq\ \mapsto\!\!\!\!\to_{\rho\sigma\delta}$.

Since the transitive closure of the parallel reduction is the $\mapsto\!\!\!\!\to_{\rho\sigma\delta}$ relation, we should show that $\Rightarrow_{\rho\sigma\delta}$ satisfies the diamond property. For this, we state some auxiliary lemmas used for showing that $\Rightarrow_{\rho\sigma\delta}$ satisfies a "strong" diamond property and we obtain as a consequence the diamond property for $\Rightarrow_{\rho\sigma\delta}$.

Lemma 3 (Permutation). *1. If $X \notin \mathsf{Fv}(C)$, then $A\{B/X\}\{C/Y\} \equiv A\{C/Y\}\{B\{C/Y\}/X\}$;*

2. If $A =_{\rho\sigma\delta} B$, then $A\{C/X\} =_{\rho\sigma\delta} B\{C/X\}$.

Proof. *By induction on the definition of substitution and $=_{\rho\sigma\delta}$ respectively.* $\square$

For the parallel reduction, the following lemma holds.

Lemma 4 (Parallel). *If $A \Rightarrow_{\rho\sigma\delta} B$ and $\vec{C} \Rightarrow_{\rho\sigma\delta} \vec{D}$, then $A\sigma_1 \Rightarrow_{\rho\sigma\delta} B\sigma_2$, with $\sigma_1 = \{\vec{C}/\vec{X}\}$, and $\sigma_2 = \{\vec{D}/\vec{X}\}$, and $\sigma_1, \sigma_2 \not\lhd \mathsf{Bv}(A)$.*

Proof. *By induction on the definition of $\Rightarrow_{\rho\sigma\delta}$.* $\square$

Lemma 5 (Diamond Property for $\Rightarrow_{\rho\sigma\delta}$). *If $A \Rightarrow_{\rho\sigma\delta} B$ and $A \Rightarrow_{\rho\sigma\delta} C$, then there exists D, such that $B \Rightarrow_{\rho\sigma\delta} D$, and $C \Rightarrow_{\rho\sigma\delta} D$.*

Indeed, we can adapt, as below, the stronger statement from [27] to our $\mathsf{P^2TS}$ and the diamond property for $\Rightarrow_{\rho\sigma\delta}$ follows immediately.

Lemma 6 (Strong Church-Rosser for $\Rightarrow_{\rho\sigma\delta}$). *If $A \Rightarrow_{\rho\sigma\delta} B$, then $B \Rightarrow_{\rho\sigma\delta} A^\diamond$.*

Proof. *By induction on the definition of $A \Rightarrow_{\rho\sigma\delta} B$:*

(Par_1)**.** *Trivial.*

(Par_2) **and** (Par_3) **and** (Par_6)**.** *We consider first case, the others being similar. We have $A_1 \wr B_1 \Rightarrow_{\rho\sigma\delta} A_2 \wr B_2$ with $A_1 \Rightarrow_{\rho\sigma\delta} A_2$, and $B_1 \Rightarrow_{\rho\sigma\delta} B_2$, and $(A_1 \wr B_1)^\diamond \equiv A_1^\diamond \wr B_1^\diamond$. By IH, $A_2 \Rightarrow_{\rho\sigma\delta} A_1^\diamond$, and $B_2 \Rightarrow_{\rho\sigma\delta} B_1^\diamond$, and thus, $A_2 \wr B_2 \Rightarrow_{\rho\sigma\delta} A_1^\diamond \wr B_1^\diamond$.*

(Par_4) **and** (Par_5)**.** *We consider the first case, the other being similar. We have therefore, $(\lambda A_1{:}\Delta_1.B_1)C_1 \Rightarrow_{\rho\sigma\delta} [A_2 \ll_{\Delta_2} C_2].B_2$, with $A_1 \Rightarrow_{\rho\sigma\delta} A_2$ and $B_1 \Rightarrow_{\rho\sigma\delta} B_2$ and $C_1 \Rightarrow_{\rho\sigma\delta} C_2$ and $\Delta_1 \Rightarrow_{\rho\sigma\delta} \Delta_2$, and $((\lambda A_1{:}\Delta_1.B_1)C_1)^\diamond \equiv [A_1^\diamond \ll_{\Delta_1^\diamond} C_1^\diamond].B_1^\diamond$. By IH, $A_2 \Rightarrow_{\rho\sigma\delta} A_1^\diamond$, and $B_2 \Rightarrow_{\rho\sigma\delta} B_1^\diamond$, and $C_2 \Rightarrow_{\rho\sigma\delta} C_1^\diamond$, and $\Delta_2 \Rightarrow_{\rho\sigma\delta} \Delta_1^\diamond$, and thus, we obtain $[A_2 \ll_{\Delta_2} C_2].B_2 \Rightarrow_{\rho\sigma\delta} [A_1^\diamond \ll_{\Delta_1^\diamond} C_1^\diamond].B_1^\diamond$.*

(Par_7)**.** *We have $[A_1 \ll_{\Delta_1} B_1].C_1 \Rightarrow_{\rho\sigma\delta} C_2\sigma_{(A_2 \prec\!\!\prec_{\emptyset}^{\Delta_2} B_2)}$ with $A_1 \Rightarrow_{\rho\sigma\delta} A_2$ and $B_1 \Rightarrow_{\rho\sigma\delta} B_2$ and $C_1 \Rightarrow_{\rho\sigma\delta} C_2$ and $\Delta_1 \Rightarrow_{\rho\sigma\delta} \Delta_2$. Since A_1 satisfies $\mathsf{ERPC}(\Delta)$ then $A_1 \equiv P\sigma_1$ with P satisfying RPC and $\sigma_1 \not\lhd \Delta$. Therefore, $A_2 \equiv P\sigma_2$ with $\sigma_1 \Rightarrow_{\rho\sigma\delta} \sigma_2$ and since $\exists\tau_2.\mathcal{A}lg(A_2 \prec\!\!\prec_{\emptyset}^{\Delta_2} B_2)$, without loss of generality, we can consider that $B_2 \equiv P\sigma_2\{\vec{D}/\vec{X}\}$ with $\vec{X} \subseteq \mathsf{Dom}(\Delta_2)$. Since $\sigma_1 \not\lhd \vec{X}$, by RPC, we obtain $B_1^\diamond \equiv P\sigma_1^\diamond\{\vec{D}^\diamond/\vec{X}\}$ and since $A_1^\diamond \equiv P\sigma^\diamond$ then, $\exists\tau^\diamond.\mathcal{A}lg(A_1^\diamond \prec\!\!\prec_{\emptyset}^{\Delta_1^\diamond} B_1^\diamond)$. Thus, $([A_1 \ll_{\Delta_1} B_1].C_1)^\diamond \equiv C_1^\diamond\tau^\diamond$. On the other hand, $\tau_2 = \{\vec{D}/\vec{X}\} \Rightarrow_{\rho\sigma\delta} \{\vec{D}^\diamond/\vec{X}\} = \tau^\diamond$ and, by IH, $C_2 \Rightarrow_{\rho\sigma\delta} C_1^\diamond$. Therefore, by Lemma 4, we can conclude that $C_2\tau_2 \Rightarrow_{\rho\sigma\delta} C_1^\diamond\tau^\diamond$.*

$\square$

Theorem 3 (Confluence). *The relation $\mapsto_{\rho\sigma\delta}$ is confluent.*

Proof. *By Lemmas 2 and 5.* $\square$

Corollary 1 (Key Lemma). *If $\Pi A_1{:}\Delta_1.B_1 =_{\rho\sigma\delta} \Pi A_2{:}\Delta_2.B_2$, then $A_1 =_{\rho\sigma\delta} A_2$, and $\Delta_1 =_{\rho\sigma\delta} \Delta_2$, and $B_1 =_{\rho\sigma\delta} B_2$.*

Proof. *Immediate from confluence.* $\square$

5.3 Subject Reduction

Subject Reduction is a standard property of PTSs that states that typing is preserved under reduction. As for confluence, Subject Reduction for $\mathsf{P^2TS}$ fails if we do not impose any condition on patterns in applications. The following simply-typed example exhibits a typable term whose reduct is not typable.

Example 5 (Spoofer). *Let*

$$\Delta \ \triangleq\ a_1{:}*, a_2{:}*, b{:}*, c{:}*, X_1{:}a_1 \triangleright b, Y_1{:}a_1$$

$$\Gamma \ \triangleq\ X_2 : a_2 \triangleright b, Y_2 : a_2, Z{:}a_1 \triangleright c$$

and consider the judgment:

$$\Gamma \vdash (\lambda(X_1\ Y_1){:}\Delta.\ (Z\ Y_1))\ (X_2\ Y_2) : c$$

The above judgment is derivable since $\Gamma, \Delta \vdash (Z\ Y_1) : c$ and $\Gamma, \Delta \vdash (X_1\ Y_1) : b$, hence by the abstraction rule $(Abst)$ we get $\Gamma \vdash (\lambda(X_1\ Y_1){:}\Delta.\ (Z\ Y_1)) : b \triangleright c$. On the other hand $\Gamma \vdash (X_2\ Y_2) : b$ so we conclude by the application rule $(Appl)$. Then $(\lambda(X_1\ Y_1){:}\Delta.\ (Z\ Y_1))\ (X_2\ Y_2)$ reduces to $Z\ Y_2$ which is not derivable in Γ.

Interestingly, the above counter-example has the same origin as the counter-example to Confluence, namely the presence of free active variables in a pattern. In particular, the pattern $X_1\ Y_1$ does not satisfy RPC *w.r.t.* Δ. Nevertheless, RPC does not immediately ensure that Subject Reduction holds for $\mathsf{P^2TS}$s, and we need to impose a suitable condition on $\mathcal{P}$ so that the solution of a well-typed matching problem is a well-typed substitution. This requirement is captured by the Subject Reduction Condition (SRC) below. Together with ERPC which is required for the Key Lemma, SRC guarantees that Subject Reduction holds.

Definition 11 (Subject Reduction Condition). *Let $\Delta \equiv X_1{:}A_1 \dots X_n{:}A_n$. The term P satisfies* SRC(Δ) *if:*

1. *If $\exists\{C_i/X_i\}^{i=1\dots n}.\mathcal{A}lg(P \lll_{\emptyset}^{\Delta} C)$, and $\Gamma, \Delta \vdash P : E$, and $\Gamma \vdash C : E$, then, for $i = 1 \dots n$,*
$$\Gamma \vdash C_i{:}A_i\{C_1/X_1, \ \dots \ , C_{i-1}/X_{i-1}\}$$
2. *If $\Gamma, \Delta \vdash P : B$ then for every $P' \in \mathcal{P}$ such that $P =_{\rho\sigma} P'$ and $\Gamma \vdash P' : B'$, we have $B =_{\rho\sigma} B'$.*

The pattern of an abstraction in P²TS should satisfy the SRC condition *w.r.t.* the domain of this abstraction. At this point of the paper, the SRC condition appears somewhat puzzling. Nevertheless, we shall later prove that the subset $\mathcal{P}$ of $\mathcal{T}$ of Definition 10 satisfies SRC. For now, we focus on the proof of Subject Reduction. The proof proceeds as with for PTSs and relies upon the following preliminary results.

Lemma 7 (Thinning). *If $\Gamma \vdash A : B$ and $\Gamma \subseteq \Delta$ and Δ legal, then $\Delta \vdash A : B$.*

Lemma 8 (Substitution). *If $\Gamma, X{:}D, \Delta \vdash A : B$, and $\Gamma \vdash C : D$, then $\Gamma, \Delta\{C/X\} \vdash A\{C/X\} : B\{C/X\}$.*

Lemma 9 (Generation).

1. *If $\Gamma \vdash s_1 : A$, then $A =_{\rho\sigma} s_2$ and $s_1 : s_2 \in \mathcal{A}$;*
2. *If $\Gamma \vdash \alpha : A$, then $\Gamma \equiv \Delta, \alpha{:}B, \Sigma$, and $A =_{\rho\sigma} B$;*
3. *If $\Gamma \vdash \lambda A{:}\Delta.B : D$, then $\Gamma, \Delta \vdash B : C$, and $\Gamma \vdash \Pi A{:}\Delta.C : s$, and $D =_{\rho\sigma} \Pi A{:}\Delta.C$;*
4. *If $\Gamma \vdash \Pi A{:}\Delta.B : D$, then $\Gamma, \Delta \vdash A : C$, and $\Gamma \vdash C : s_1$, and $\Gamma, \Delta \vdash B : s_2$, and $(s_1, s_2, s_3) \in \mathcal{R}$, and $D =_{\rho\sigma} s_3$;*
5. *If $\Gamma \vdash A\, B : F$, then $\Gamma \vdash A : \Pi C{:}\Delta.D$, and $\Gamma \vdash B : E$, and $\Gamma, \Delta \vdash C : E$, and $F =_{\rho\sigma} [C \ll_\Delta B].D$;*
6. *If $\Gamma \vdash [C \ll_\Delta B].A : F$, then $\Gamma, \Delta \vdash C : D$ and $\Gamma \vdash B : D$ and $\Gamma, \Delta \vdash A : E$ and $F =_{\rho\sigma} [C \ll_\Delta B]^\top.E$;*
7. *If $\Gamma \vdash A, B : C$, then $\Gamma \vdash A : D$ and $\Gamma \vdash B : D$ and $C =_{\rho\sigma} D$;*

Lemma 10 (Correctness of Types). *If $\Gamma \vdash A : B$ then $\Gamma \vdash B : C$ or $B \in \mathcal{S}^\top$.*

The last result needed for proving Subject Reduction is the Context Conversion Lemma, which states that convertible contexts type the same judgments.

Lemma 11 (Context Conversion). *If $\Gamma \vdash A : B$ and Δ is legal and $\Gamma =_{\rho\sigma} \Delta$ then $\Delta \vdash A : B$.*

Proof. *One first prove by induction on the structure of derivations the following: if $\Gamma \vdash A : B$ and $\Delta \vdash X : A$ for every $(X{:}A) \in \Gamma$, then $\Delta \vdash A : B$. Then one proceeds by induction on the length of Γ.* □

Theorem 4 (Subject Reduction). *If $\Gamma \vdash A : B$, and $A \mapsto\!\!\!\!\to_{\rho\sigma} C$, then $\Gamma \vdash C : B$.*

Proof. *By induction on the structure of A. We only treat the cases where A is a ρ-redex or σ-redex; all other cases follow easily.*

(ρ) Assume that $(\lambda A{:}\Delta.B)\, C \to_\rho [A \ll_\Delta C].B$ and that $\Gamma \vdash (\lambda A{:}\Delta.B)\, C : D$.
By successive applications of Generation, $\Gamma, \Delta' \vdash A' : F'$, and $\Gamma \vdash C : F'$, and $\Gamma, \Delta \vdash B{:}E$, and $\Gamma, \Delta \vdash A{:}F$, and $\Gamma \vdash F{:}s$, and $D =_{\rho\sigma} [A' \ll_{\Delta'} C].E'$, and $\Pi A{:}\Delta.E =_{\rho\sigma} \Pi A'{:}\Delta'.E'$. By the Key Lemma, we have $A =_{\rho\sigma} A'$, and $\Delta =_{\rho\sigma} \Delta'$, and $E =_{\rho\sigma} E'$. By Context Conversion, we have $\Gamma, \Delta \vdash A'{:}F'$, and by SRC*(2), we have $F =_{\rho\sigma} F'$. By (Conv), we have $\Gamma \vdash C{:}F$. By (Subst), $\Gamma \vdash [A \ll_\Delta C].B{:}[A \ll_\Delta C]^\top.E$. Furthermore $E \notin \mathcal{S}^\top$, hence $[A \ll_\Delta C]^\top.E \equiv [A \ll_\Delta C].E$, and by Correctness of Types, we have $\Gamma, \Delta \vdash E{:}G$. Hence by (Subst), $\Gamma \vdash [A \ll_\Delta C].E{:}[A \ll_\Delta C]^\top.G$, and hence by (Conv), we have $\Gamma \vdash [A \ll_\Delta C].B{:}D$ as desired.*

(σ) Assume that $[A \ll_\Delta C].B \to_\sigma B\sigma$ with $\exists\sigma.\mathcal{A}lg(A \lll_{\emptyset}^{\Delta} C)$, and that $\Gamma \vdash [A \ll_\Delta C].B : D$.
By Generation, we have $\Gamma, \Delta \vdash A : E$, and $\Gamma \vdash C : E$, and $\Gamma, \Delta \vdash B : F$, and $D =_{\rho\sigma} [A \ll_\Delta C]^\top.F$. By SRC*(1), we have $\Gamma \vdash B\sigma : F\sigma$. By Correctness of Types, $\Gamma \vdash F : G$ or $F \in \mathcal{S}^\top$. In the first case, we apply (Subst) to get $\Gamma \vdash [A \ll_\Delta C].F : [A \ll_\Delta C]^\top.G$ and we conclude by (Conv). In the second case, we get $D =_{\rho\sigma} F$, hence $D \mapsto\!\!\!\!\to_{\rho\sigma} F$ by Confluence; therefore, F is a subterm of D. By a simple analysis of the typing rules, it follows that $D \equiv F$ and we are done.* □

We now turn to the characterization of SRC. We start with a preliminary result.

Lemma 12 (Unicity of Typing). *If $\Gamma \vdash A : B$ and $\Gamma \vdash A : C$, then $B =_{\rho\sigma} C$.*

Proof. *By induction on the structure of derivations, using the functionality of specifications.* □

Lemma 13 (Characterization of SRC(Δ)). *If $P \in \mathcal{P}$ and $\sigma \not\ll \Delta$ then $P\sigma$ satisfies* SRC(Δ).

Proof. *By induction on P.* □

5.4 P²TS as Logics

Many type systems and logics, including the systems of Barendregt's λ-cube [4], can be cast in the framework of PTSs. In perfect symmetry, the framework of P²TSs allows us to formulate a matching-based version of many type systems. In the case of the λ-cube, its matching-based counterpart, which we call *$\lambda^{\ll}$-cube with matching* ($\lambda^{\ll}$-cube for short), is depicted in Figure 6, where $\mathcal{S} = \{*, \square\}$, and $\mathcal{A} = \{(*, \square)\}$; note that we use (s_1, s_2) to denote rules of the form (s_1, s_2, s_2).

PTSs, and in particular the systems of the λ-cube, have a well-understood logical theory via the Curry-Howard Isomorphism and the (impredicative) encoding of data-types in type systems, and are used as the foundations of proof-assistants. A detailed analysis of the logical status of P²TSs, including an extension of the Curry-Howard Isomorphism, is left for future work, but we provide two elementary results that establish that P²TSs are logically sound.

First, we show that P²TSs are a conservative extension of PTSs. Indeed, it is possible to embed every PTS into its

System	Rules			
$\lambda^{\ll}\!\rightarrow$	$(*,*)$			
$\lambda^{\ll}2$	$(*,*)$	$(\Box,*)$		
$\lambda^{\ll}\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda^{\ll}\omega$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda^{\ll}P$	$(*,*)$		$(*,\Box)$	
$\lambda^{\ll}P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda^{\ll}P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda^{\ll}P\omega$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

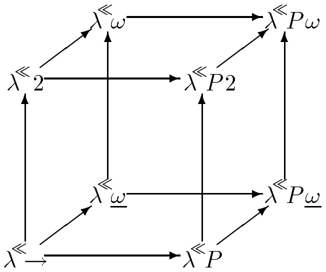

Figure 6: The $\lambda^{\ll}$-cube

corresponding $\mathsf{P^2TS}$ using the (trivial) translation $\dagger$

$$a^\dagger \triangleq a \qquad (\surd X{:}A.B)^\dagger \triangleq \surd X{:}(X{:}A^\dagger).B^\dagger$$
$$X^\dagger \triangleq X \qquad (A\ B)^\dagger \triangleq A^\dagger\ B^\dagger$$

THEOREM 5 (CONSERVATIVITY). *$\mathsf{P^2TS}$s are a conservative extension of PTSs in the sense that for every PTS pseudo-context Γ and PTS pseudo-terms A and B*

$$\Gamma \vdash_{\mathsf{PTS}} A : B \iff \Gamma^\dagger \vdash_{\mathsf{P^2TS}} A^\dagger : B^\dagger$$

PROOF. *The direct implication is trivial. For the reverse implication, we prove by induction on the structure of derivations that $\Gamma^\dagger \vdash_{\mathsf{P^2TS}} A^\dagger : C$ implies $\Gamma \vdash_{\mathsf{PTS}} A : B$ for some B such that $B^\dagger =_{\rho\sigma\delta} C$.* $\Box$

Second, we show that normalizing $\mathsf{P^2TS}$s are consistent. The proof relies upon the following observation.

LEMMA 14 (CHAINING). *If A, B, Δ are in normal form and $\Gamma \vdash ([A \ll_\Delta B].C)\ D : E$, then $\exists\sigma.Alg(A \prec\!\!\prec_{\emptyset}^{\Delta} B)$.*

PROOF. *Apply Generation twice.* $\Box$

THEOREM 6 (CONSISTENCY IN $\mathsf{P^2TS}$). *Any normalizing $\mathsf{P^2TS}$ is logically consistent, i.e. for every sort $s \in \mathcal{S}$, $X{:}s \not\vdash A : X$.*

PROOF. *The proof follows [4]: assume without loss of generality that A is in normal form, and proceed by analysis on the shape of normal forms, using the Chaining Lemma to rule out the case $A \equiv ([A_1 \ll_\Delta A_2].A_3)\ A_4\ \ldots\ A_n$.* $\Box$

6. CONCLUSION

$\mathsf{P^2TS}$s provide a conservative extension of PTSs with matching, and enjoy all elementary properties of PTSs. The next question on our agenda is to determine whether $\mathsf{P^2TS}$s also enjoy non-elementary properties of PTSs:

- *w.r.t.* strong normalization, we conjecture that standard model construction techniques can be used to prove strong normalization of the $\lambda^{\ll}$-cube;
- *w.r.t.* type checking/inference, we conjecture that existing algorithms for PTSs adapt readily to $\mathsf{P^2TS}$s.

Besides, we would like to enhance the expressive and computational power of $\mathsf{P^2TS}$s, so as to provide support for:

- handling failures/exceptions by introducing a uniform notion of failure Null, which records failures but discards their explanation, and its corresponding conditional rewrite rule $A\ \mathsf{Cond}\ B$ implies $[A \ll_\Delta B].C \mapsto_{\rho\sigma\delta} \mathsf{Null}$ for some suitable condition Cond;
- matching modulo an equational theory, as in the rewriting calculus [8, 9], upon which $\mathsf{P^2TS}$s are built, and which takes as a parameter an equational theory modulo which matching is performed. In a similar vein, it would be interesting to study $\mathsf{P^2TS}$s with a limited form of decidable higher-order unification, in the style of λ-Prolog [21, 22, 26];
- encoding dependent case analysis, pattern-matching à la Coquand and algebraic type systems. In view of the Chaining Lemma, $\mathsf{P^2TS}$s cannot code dependent case analysis, and hence cannot provide a foundation for pattern matching in dependent type theory. It would be interesting to investigate whether endorsing more powerful rules for structures and a subtyping calculus derived from matching would suffice to encode dependent case analysis;
- explicit substitutions. The extension is not trivial, because of delayed matching constraints, but the resulting formalism could serve as the core *engine* of a little type-checker underneath of a powerful proof assistant.

We conclude with a challenge for future work: *Extending the Curry-Howard Isomorphism.* The extension can be considered from the point of view of sequent calculi, deduction modulo, and natural deduction respectively. From the point of view of sequent calculi, it remains to investigate how $\mathsf{P^2TS}$s can be used to extend previous results on term calculi for sequent calculi, and how their extension with matching theories can be used to provide suitable term calculi for deduction modulo. From the point of view of natural deduction, $\mathsf{P^2TS}$s correspond to an extension of natural deduction where parts of proof trees are discharged instead of assumptions. To our best knowledge, such an extended form of natural deduction has not been considered previously, but it seems interesting to investigate whether such an extended natural deduction could find some applications in proof assistants, *e.g.* for transforming and optimizing proofs.

Acknowledgments. The authors are sincerely grateful to all anonymous referees for their extremely useful comments, and to one particular referee for pointing out some subtleties in the interaction of patterns, dependent types and polymorphism. They also wish to thank Vincent van Oostrom, Pierre Courtieu, Jöelle Despeyroux, Philippe de Groote for fruitful discussions and comments. Finally, Luigi would like to thank Simonetta Ronchi della Rocca and Furio Honsell, for the time spent to teach to him the fundamentals and the insight of the "cubes-stuff".

7. REFERENCES

[1] L. Augustsson. Cayenne: A Language with Dependent Types. In *Proc. of ICFP*, pages 239–250. ACM Press, 1998.

[2] F. Barbanera, M. Fernández, and H. Geuvers. Modularity of Strong Normalisation and Confluence in the Algebraic λ-Cube. *Journal of Functional Programming*, 7(6):613–660, 1997.

[3] H. Barendregt. *Lambda Calculus: its Syntax and Semantics*. North Holland, 1984.

[4] H. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, volume II, pages 118–310. Oxford University Press, 1992.

[5] H. Barendregt and H. Geuvers. Proof Assistants Using Dependent Type Systems. In *Handbook of Automated Reasoning*, volume II, chapter 18, pages 1149–1238. Elsevier Publishing, 2001.

[6] G. Barthe and T. Coquand. An Introduction to Dependent Type Theory. In *Proc. of Applied Semantics Summer School*, volume 2395 of *LNCS*. Springer-Verlag, 2002.

[7] F. Blanqui. *Type Theory and Rewriting*. PhD thesis, Université de Paris-Sud, 2001.

[8] H. Cirstea and C. Kirchner. The Rewriting Calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, 2001.

[9] H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In *Proc. of RTA*, volume 2051 of *LNCS*, pages 77–92. Springer-Verlag, 2001.

[10] H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. In *Proc. of FOSSACS*, volume 2030 of *LNCS*, pages 166–180, 2001.

[11] H. Cirstea, C. Kirchner, and L. Liquori. Rewriting Calculus with(out) Types. In *Proc. of WRLA*, volume 71 of *ENTCS*, 2002.

[12] T. Coquand. Pattern Matching with Dependent Types. In *Proc. of Logical Frameworks*, pages 66–79, 1992.

[13] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via Explicit Substitutions: The Case of Higher-Order Patterns. In *Proc. of JICSLP*. MIT Press, 1996.

[14] G. Faure and C. Kirchner. Exceptions in the Rewriting Calculus. In *Proc. of RTA*, volume 2378 of *LNCS*, pages 66–82. Springer-Verlag, 2002.

[15] H. Geuvers and M.J. Nederhof. A Modular Proof of Strong Normalisation for the Calculus of Constructions. *Journal of Functional Programming*, 1(2):155–189, 1991.

[16] G. Huet. *Résolution d'Equations dans les Langages d'Ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7, 1976.

[17] J.P. Jouannaud and M. Okada. Abstract Data Type Systems. *Theoretical Computer Science*, 173(2):349–391, 1997.

[18] D. Kesner, L. Puel, and V. Tannen. A Typed Pattern Calculus. *Information and Computation*, 124(1):32–61, 1996.

[19] J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, 1980. PhD Thesis.

[20] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory Reduction Systems: Introduction and Survey. *Theoretical Computer Science*, 121(1&2):279–308, 1993.

[21] D. Miller. A Logic Programming Language with Lambda-abstraction, Function Variables, and Simple Unification. In *Proc. of ELP*, volume 475 of *LNCS*, pages 253–281. Springer-Verlag, 1991.

[22] D. Miller, G. Nadathur, F. Pfenning, and A. Shedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logics*, 51(1-2):125–157, 1991.

[23] T. Nipkow and C. Prehofer. Higher-order Rewriting and Equational Reasoning. In *Automated Deduction — A Basis for Applications. Volume I: Foundations*. Kluwer, 1998.

[24] S. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, 1987.

[25] S.L. Peyton Jones and E. Meijer. Henk: a Typed Intermediate Language. In *Types in Compilation Workshop*, 1997.

[26] C. Schürmann. *Automating the Meta-Theory of Deductive Systems*. Phd thesis, Carnegie-Mellon University, 2000.

[27] M. Takahashi. Parallel Reductions in λ-calculus. *Journal of Symbolic Computation*, 7(2):113–123, 1989.

[28] S. van Bakel, L. Liquori, S. Ronchi della Rocca, and P. Urzyczyn. Comparing Cubes of Typed and Type Assignment System. *Annals of Pure and Applied Logics*, 86(3):267–303, 1997.

[29] V. van Oostrom. Lambda Calculus with Patterns. Technical Report IR-228, Faculteit der Wiskunde en Informatica, Vrije Universiteit Amsterdam, 1990.

[30] D. A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.