A Language for Verification and Manipulation of Web Documents

(Extended Abstract)

Luigi Liquori

INRIA, France

Furio Honsell

DIMI, University of Udine, Italy

Rekha Redamalla

Birla Science Center, Adarsh Nagar, Hyderabad, India

Abstract

In this paper we develop the language theory underpinning the logical framework PLF. This language features lambda abstraction with patterns and application via pattern-matching. Reductions are allowed in patterns. The framework is particularly suited as a metalanguage for encoding rewriting logics and logical systems where proof terms have a special syntactic constraints, as in term rewriting systems, and rule-based languages. PLF is a conservative extension of the well-known Edinburgh Logical Framework LF. Because of sophisticated pattern matching facilities PLF is suitable for verification and manipulation of HXML documents.

1 Introduction

History

Since the introduction of Logical Frameworks [6, 9], blending dependent typed λ-calculi with rewriting systems has been a major challenge in the last decades, see [16, 11, 18, 5, 7, 13]. Blending lambda calculi and rewrite rules enhances the usability of the system as a metalanguage for logics. The expressiveness of the system does not increase, since already the Logical Framework of [9] is a universal language for encoding formal systems. Clearly rewrite systems can provide in many instances much more natural and transparent encodings, thus improving the overall pragmatic usability of the system, and

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

the possibility of automating decision procedures, such as checking and encoding equality.

The Pattern Logical Framework PLF

In this paper, we present an uniform framework based on a dependent typed lambda calculus enriched with pattern matching in lambda abstraction, called PLF. In contrast to the simple λ-calculus, the pattern-matching algorithm can either fire a substitution, or stick the computation unless further substitution are provided. The following simple example illustrates the point: $M \equiv (\lambda(f\,y).y)\,x$ is stuck, but

$$(\lambda(f\,x).M)\,(f\,(f\,3)) \mapsto\!\!\!\!\to_{\beta} 3$$

Furthermore, for a given pattern inside an abstraction, the user can explicitly discriminate between variables that will be bound in the body, and variables that can be bound in an external context containing the abstraction itself. This freedom is particularly appreciated in patterns that evolve (by reduction or substitution) during execution, *e.g.*

$$(\lambda x.\lambda P[x].N)\,M \mapsto_{\beta} \lambda P[M].N$$

PLF extends the subsystem of [3] corresponding to LF in allowing reductions inside patterns. As it is well-known, since the seminal work of [18], variables in patterns can be bound only if they occur linearly and not actively (*i.e.* not in functional position), hence extra care has to be payed when reductions are allowed in patterns.

PLF as an untyped rule-based language

The first step to appreciate the framework is to play with the untyped Rewriting calculus (RHO) [17]; is essentially the untyped *alter ego* of our PLF. Thanks to its high-level pattern-matching features, the RHO is particularly well-suitable for programming various transformations on HXML documents (traversals, trees-transformations, etc.). Its design follows a longstanding research on rule-based languages, like Elan, Maude, OBJ*, ASF+SDF. Programming by pattern-matching in the RHO allows us to implement a large collection of rule-based systems and algorithms to manipulate or simply browse HXML documents. Most of the systems presented in the workshop, like the one of Ballis *et al.* [2, 1], or the one of Kirchner *et al.* [12], or the one of Lucas [15] *are* rule-based systems, hence *de facto* they can be expressed in the Rewriting calculus. Also more complex systems employed to check legacy of HXML code, like the one of Finkelstein [8], are "programmable" in the RHO. The following simple example, written in the running syntax of the untyped PLF interpreter [14], illustrates a function that recursively "navigates" web pages trying to index a given page (in the following read a rewrite rule `P->M` as an untyped PLF term, *i.e.* λ`P.M`, and let any sequence of capital letters be a variable, and let any other sequence be an algebraic constant):

```
LINK   = rho.inria.fr;;                                              http link to find
INDEX  = L -> (ohyeahpleaseindex L);;                          simple function indexing L
LOAD   = (gigi.inria.fr -> GIGI,                              function loading http pages
          claude.inria.fr -> CLAU,
         horatiu.inria.fr -> HORA)
                                                                          four http pages
WWV  = (html ((markup (markup pc),(href claude.inria.fr)),(markup (markup pc),(markup pc))));;
CLAU = (html ((markup (markup pc),(markup pc)),(markup (href horatiu.inria.fr),(markup pc))));;
HORA = (html ((markup (href gigi.inria.fr),(markup pc)),(markup (markup pc),(markup pc))));;
GIGI = (html ((markup ((markup pc),(markup pc))),(markup ((markup pc),(href rho.inria.fr)))));;
[FINDLINK = (html BODY) -> (FIND BODY)                 find a LINK in a web page (recursively)
 |
 FIND = ( pc                  ->|,                                          the core function
          (markup X)          -> (FIND X) ,
          (href LINK)         -> (INDEX LINK),
          (href L\LINK)       -> (FINDLINK (LOAD L)),
          ((href LINK)  ,BODY) -> (INDEX LINK),
          ((href L\LINK),BODY) -> ((FINDLINK (LOAD L)),(FIND BODY)),
          ((markup X)   ,BODY) -> ((FIND X)           ,(FIND BODY)))
];;
(FINDLINK WWV);;                                              calling FINDLINK on input WWV
```

In the rest of the paper, we show some classical HXML-oriented routines, written in the RHO, that can be runned (in the "untyped' mode") and certified in our PLF (using dependent type theory).

PLF as a type-based & pattern-matching based metalanguage

More interesting, the full framework features formal reasoning on the semantics of the routines you have programmed. This can be done by using meta-programming and dependent-types in PLF; in other words we can reason about the meaning of an HXML routine and prove properties like the one proving that *the routine meets the specification*, *i.e.* "it really does the job it was conceived for".

We present one crucial example of encoding, which capitalizes on patterns, namely we completely formalize in the framework the operational semantics of the Rewriting calculus (RHO). We formalize syntax and small-step operational semantics. The main advantages of using PLF instead of the original Edinburgh Logical Framework LF is that PLF is equipped with built-in pattern-matching facilities (a superset of the pattern-matching facilities of the Rewriting Calculus). This permits to specify the target language using less subcategories and, as a direct consequence, all proofs, like the crucial one of Church Rosser becomes simpler in size and in complexity, since proofs terms are smaller w.r.t. the equivalent ones in plain LF. This is one achievement of the framework. A (functional) certified interpreter can be also extracted from the specifications.

Sum up

Although the first part of the paper presenting PLF is "theoretically" dense (the addiction of pattern-matching and rewriting-based features to the Logical Framework is not so simple as one might imagine) we think that the second part is more "easy to digest" for an HXML-acolyte. At the very end of the

story, programming and verifying HXML-oriented routines (and their interpreters) within the rule-based paradigm, using high-level pattern-matching features offered by the Pattern Logical Framework (PLF), and running those routines in the corresponding untyped Framework (RHO) is one of the main challenges of this research vein and is it our humble opinion that this paper completely fits the scope of the workshop.

Summarizing, PLF and its *alter ego*, the RHO adhere to the novel fundamental paradigm of *programming with proofs*, with a particular increasingly interest on web applications.

The full meta-theory of PLF will appear in a companion paper [10].

2 The Pattern Logical Framework

We present the syntax of PLF, a logical framework with pattern-matching features. Since patterns occurs as *binders* in abstractions, the types of the "matchable" variable in the pattern are declared in suitable *contexts*, *i.e.* a pattern abstraction has the form $\lambda P{:}\Delta.M$, where Δ contains the types of the free-variables of P (bound in M). Therefore, the contexts defining the types of the free variables of these patterns are given explicitly as part of the abstraction. The context Δ can discriminate on variables that are suitable to be matched and variables that are not, the latter are *de facto* considered as constants. This treatment of constants simplifies the presentation of the system since constants are now considered as variables that cannot be bound. This mechanism implies also that some patterns can evolve during reduction via substitution; to do this in a sound way, the pair pattern-context must be constrained to satisfy suitable restrictions.

2.1 PLF*'s Terms*

The first definition introduces the pseudo-syntax for kinds, families, objects and contexts.

Definition 2.1 [PLF's Pseudo-syntax]

$$\Gamma, \Delta \in \mathcal{C} \qquad \Gamma \ ::= \emptyset \mid \Gamma, x{:}K \mid \Gamma, x{:}A$$

$$K \in \mathcal{K} \qquad K \ ::= \mathsf{Type} \mid \Pi M{:}\Delta.K$$

$$A, B, C, P, Q \in \mathcal{F} \qquad A \ ::= x \mid \Pi M{:}\Delta.A \mid \lambda M{:}\Delta.A \mid A\, M$$

$$M, N, P, Q \in \mathcal{O} \qquad M \ ::= x \mid \lambda M{:}\Delta.M \mid M\, M$$

As usual, application associates to the right. Let "T" range over any term in the calculus, and let the symbol "$\checkmark$" range over the set $\{\lambda, \Pi\}$. To ease the notation, we write $\checkmark x{:}T_1.T_2$ for $\checkmark x{:}(x{:}T_1).T_2$ in case of a variable-pattern.

Intuitively, the context Δ in $\lambda P{:}\Delta.M$ contains the type declarations of some (but not all) of the free variables appearing in the pattern P. These variables are bound in the (pattern and body of the) abstraction; the reduction of an application $(\lambda P{:}\Delta.M)\,N$ strongly depends on the free-variables of P declared in Δ, all the other variables being handled as constants. The free variables of P not declared in Δ are not bound in M but can be bound outside the scope of the abstraction itself. For example, in the abstraction

$$\lambda(x\,y\,z){:}(y{:}w, z{:}w).M$$

the y and z variables (of type w) are bound in M, while the x variable (not declared in the context) is considered as free (*i.e.* it is *de facto* handled as a constant). As in ordinary systems dealing with dependent types, we suppose that in the context $\Gamma, x{:}T$, the variable x does not appear free in Γ, and T. Some remarks to understand this syntax are in order. The following predicates characterize the set of *legal/good* patterns w.r.t. a given context.

Definition 2.2 [Safe Patterns]
A safe pattern is characterized by having no free variables in functional position, and by its linearity. This can be carefully formalized, but for the purpose of this paper we just write

$$\mathsf{SPC}(P;\mathbb{V}) \triangleq \neg\mathsf{APC}(P;\mathbb{V}) \wedge \mathsf{LPC}(P;\mathbb{V}).$$

where $P \in \mathcal{O}$, and $\mathbb{V}$ is a finite set of variables. This definition can be extended point-wise to substitutions, families and kinds.

Given the above restrictions on objects occurring in patterns, we can thus define a *safe/legal* PLF *syntax* as follows:

Definition 2.3 [PLF's Safe Syntax]
A term in the PLF syntax is safe if any subterm $\checkmark P{:}\Delta.T$ occurring in it is such that

$$\mathsf{SPC}(P;\mathsf{Dom}(\Delta)) \wedge \mathsf{SPC}(T;\mathsf{Dom}(\Delta))$$

In the rest of the paper we shall consider only safe terms. The definition of free variables needs to be customized as follows.

Definition 2.4 [Free Variables]
The set Fv of free variables is given by:

$$\mathsf{Fv}(\checkmark T_1{:}\Delta.T_2) \triangleq (\mathsf{Fv}(T_1) \cup \mathsf{Fv}(T_2) \cup \mathsf{Fv}(\Delta)) \setminus \mathsf{Dom}(\Delta)$$

$$\mathsf{Fv}(\Delta, x{:}T) \triangleq \mathsf{Fv}(\Delta) \cup (\mathsf{Fv}(T) \setminus \mathsf{Dom}(\Delta))$$

The set Bv of bound variables of a term is the set of variables in the term which are not free. Since we work modulo α-conversion, we suppose

that all bound variables of a term have different names and therefore, the domains of all contexts are distinct. A suitable, intuitive, (re)definition of simultaneous substitution application (denoted by θ) to deal with the new forms of abstraction is assumed.

2.2 Operational Semantics

PLF features pattern abstractions whose application requires solving matching problems. The matching algorithm is first-order, terminating, deterministic, and works modulo α-conversion and Barendregt's hygiene-convention. We write θ for the successful output of $\mathcal{A}lg(P \cdot N \cdot \mathbb{V})$. It is presented in [10]. The next definition introduces the classical notions of one-step, many-steps, and congruence relation of $\rightarrow_\beta$.

Definition 2.5 [One/Many-Steps, Congruence]
Let $\theta = \mathcal{A}lg(P \cdot N \cdot \mathsf{Dom}(\Delta))$.

(i) The top-level rules are

$$(\beta_{\mathsf{p}}\text{−Obj}) \quad (\lambda P{:}\Delta.M)\, N \rightarrow_\beta M\theta$$

$$(\beta_{\mathsf{p}}\text{−Fam}) \quad (\lambda P{:}\Delta.A)\, N \rightarrow_\beta A\theta$$

(ii) Let $\mathsf{C}[-]$ denote a pseudo-context with a "single hole" inside, it is defined as follows

$$\mathsf{C}[-] ::= [-] \mid \mathsf{C}[-]\,T \mid T\,\mathsf{C}[-] \mid \checkmark P{:}\Delta.\mathsf{C}[-] \mid \checkmark P{:}\mathsf{C}[-].T \mid \Gamma, x{:}\mathsf{C}[-]$$

and let $\mathsf{C}[T]$ be the result of filling the hole with the term T. The one-step evaluation $\mapsto_\beta$ is defined by the following inference rules

$$\frac{T_1 \rightarrow_\beta T_2}{\mathsf{C}[T_1] \mapsto_\beta \mathsf{C}[T_2]}\,(\mathsf{Ctx}) \qquad \frac{\begin{array}{ll} P \rightarrow_\beta Q & \mathsf{SPC}(Q; \mathsf{Dom}(\Delta)) \\ \mathsf{Fv}(P) \cap \mathsf{Dom}(\Delta) = \mathsf{Fv}(Q) \cap \mathsf{Dom}(\Delta) & \end{array}}{\checkmark P{:}\Delta.T \mapsto_\beta \checkmark Q{:}\Delta.T}\,(\mathsf{Ctx}^{\checkmark})$$

The intended meaning of the (Ctx) rule is the usual one. Rule ($\mathsf{Ctx}^{\checkmark}$) forbids $\mathsf{K}\beta$-reductions in patterns enforces the safe pattern condition in both redex and the reduced term.

(iii) The many-step evaluation $\mapsto\!\!\!\!\rightarrow_{\beta_{\mathsf{p}}}$ and the congruence relation $=_\beta$ are respectively defined as the reflexive-transitive and reflexive-symmetric-transitive closure of $\mapsto_{\beta_{\mathsf{p}}}$.

Dependent Syntax

$$\Gamma ::= \emptyset \mid \Gamma, x{:}K \mid \Gamma, x{:}A$$

$$K ::= \mathsf{Type} \mid \Pi M{:}\Delta.K$$

$$A ::= x \mid \Pi M{:}\Delta.A \mid \lambda M{:}\Delta.A \mid A\,M$$

$$M ::= x \mid \lambda M{:}\Delta.M \mid M\,M$$

Safe Syntax

$$\checkmark M{:}\Delta.T \in \mathcal{C}, \mathcal{K}, \mathcal{F}, \mathcal{O} \implies \mathsf{SPC}(M;\Delta) \wedge \mathsf{SPC}(T;\Delta)$$

Contexts rules

$$\frac{}{\vdash \emptyset}\ (\mathsf{C{\cdot}Empty})$$

$$\frac{\vdash \Gamma \quad \Gamma \vdash K \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash \Gamma, x{:}K}\ (\mathsf{C{\cdot}Kind})$$

$$\frac{\vdash \Gamma \quad \Gamma \vdash A : \mathsf{Type} \quad x \notin \mathsf{Dom}(\Gamma)}{\vdash \Gamma, x{:}A}\ (\mathsf{C{\cdot}Type})$$

Kind rules

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Type}}\ (\mathsf{K{\cdot}Type})$$

$$\frac{\Gamma, \Delta \vdash M : A \quad \Gamma, \Delta \vdash K}{\Gamma \vdash \Pi M{:}\Delta.K}\ (\mathsf{K{\cdot}Pi})$$

Families rules

$$\frac{\vdash \Gamma \quad x{:}K \in \Gamma}{\Gamma \vdash x : K}\ (\mathsf{F{\cdot}Var})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash A : \mathsf{Type}}{\Gamma \vdash \Pi M{:}\Delta.A : \mathsf{Type}}\ (\mathsf{F{\cdot}Pi})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash A : K}{\Gamma \vdash \lambda M{:}\Delta.A : \Pi M{:}\Delta.K}\ (\mathsf{F{\cdot}Abs})$$

$$\frac{\Gamma \vdash A : \Pi N{:}\Delta.K \quad \Gamma, \Delta \vdash N : B \quad \Gamma \vdash M : B}{\Gamma \vdash A\,M : (\lambda N{:}\Delta.K)\,M}\ (\mathsf{F{\cdot}Appl})$$

$$\frac{\Gamma \vdash A : K' \quad \Gamma \vdash K \quad K =_\beta K'}{\Gamma \vdash A : K}\ (\mathsf{F{\cdot}Conv})$$

Object rules

$$\frac{\vdash \Gamma \quad x{:}A \in \Gamma}{\Gamma \vdash x : A}\ (\mathsf{O{\cdot}Var})$$

$$\frac{\Gamma, \Delta \vdash M : B \quad \Gamma, \Delta \vdash N : A}{\Gamma \vdash \lambda M{:}\Delta.N : \Pi M{:}\Delta.A}\ (\mathsf{O{\cdot}Abs})$$

$$\frac{\Gamma \vdash M : \Pi P{:}\Delta.A \quad \Gamma, \Delta \vdash P : B \quad \Gamma \vdash N : B}{\Gamma \vdash M\,N : (\lambda P{:}\Delta.A)\,N}\ (\mathsf{O{\cdot}Appl})$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \mathsf{Type} \quad A =_\beta B}{\Gamma \vdash M : B}\ (\mathsf{O{\cdot}Conv})$$

Figure 1. PLF's Type Rules

2.3 PLF*'s Type System*

PLF involves type judgments of the following shape:

$$\vdash \Gamma \qquad \Gamma \vdash K \qquad \Gamma \vdash A : \mathsf{Type} \quad \Gamma \vdash M : A$$

In the type system rules of PLF (Figure 1) some rules are quite similar to the ones of the classical logical framework, but others deserve a brief explanation:

- The $(\mathsf{F{\cdot}Abs}), (\mathsf{O{\cdot}Abs})$ rules deal with λ-abstractions in which we bind over (non trivial) patterns; this rule requires that the pattern and the body of the abstraction are typable in the extended context Γ, Δ.
- The $(\mathsf{F{\cdot}Appl}), (\mathsf{O{\cdot}Appl})$ rules, give a type to an application; this type contrasts with classical type systems which utilize meta-substitutions. Suitable applications of $(\mathsf{F{\cdot}Conv}), (\mathsf{O{\cdot}Conv})$ rule can reduce this type.
- The $(\mathsf{K{\cdot}Pi}), (\mathsf{F{\cdot}Pi})$ rules, give a type to a kind and family products. As for λ-abstractions we require that the pattern and the body of the abstraction are typable in the extended context Γ, Δ.

Syntax and Operational Semantics

$P ::= x \mid A \mid P \twoheadrightarrow M \qquad (P \twoheadrightarrow M)\, N \;\to_\rho\; M\theta \quad \text{with } \theta = \mathcal{A}lg(P \cdot N \cdot \mathsf{Fv}(P))$

$A ::= f \mid A\, P \qquad (M_1\,, M_2)\, M_3 \;\to_\delta\; (M_1\, M_3\,, M_2\, M_3)$

$M ::= P \mid M\, M \qquad$ N.B. both $(M_1\,, M_2)$ and $\to_\delta$ are derivable

Syntactic Categories

o : Type

Operations (we short o^n for $\underbrace{o \to \ldots \to o}_{n \text{ times}}$ and $\checkmark\mathsf{C}[\overline{x}]{:}(\overline{x{:}o^n})$ for $\checkmark\mathsf{C}[\overline{x^{o^n}}]$)

$\mathsf{Alg} : o^2 \qquad \mathsf{Rew} : o^2 \to o \qquad \mathsf{App} : o^3 \qquad \mathsf{Pair} : o^3$

Judgments

$= \quad : o \to o \to \mathsf{Type}$

Rewriting encoding $[\![-]\!] : \mathsf{Rho} \Rightarrow \mathsf{PLF}$

$[\![x]\!] \triangleq x \qquad [\![P \twoheadrightarrow M]\!] \triangleq \mathsf{Rew}\ (\lambda [\![P]\!]{:}\Delta.\ [\![M]\!]) \quad \Delta \triangleq \overline{\mathsf{Fv}(P){:}o}$

$[\![f]\!] \triangleq \mathsf{Alg}\ f \qquad [\![M\, N]\!] \triangleq \mathsf{App}\ [\![M]\!]\, [\![N]\!]$

$[\![A\, P]\!] \triangleq \mathsf{App}\ [\![A]\!]\, [\![P]\!] \qquad [\![M\,, N]\!] \triangleq \mathsf{Rew}\ (\lambda x{:}o.\ \mathsf{Pair}\ (\mathsf{App}\ [\![M]\!]\, x)\, (\mathsf{App}\ [\![N]\!]\, x))$

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}}$ $\mathsf{Eq}_{\mathsf{symm}}$ $\mathsf{Eq}_{\mathsf{trans}}$ $\mathsf{Eq}_{\mathsf{ctx}}$ as in the lambda calculus encoding (see [9])

$\mathsf{Rho} : \Pi r{:}o^2.\ \Pi a{:}o.\ \mathsf{App}\ (\mathsf{Rew}\ r)\, a = r\, a$

$\mathsf{Eta} : \Pi x{:}o.\ \mathsf{Rew}\ (\lambda y{:}o.\ \mathsf{App}\ x\, y) = x$

$\mathsf{Xi} : \Pi r{:}o^2.\ \Pi s{:}o^2.\ (\Pi a{:}o.\ r\, a = s\, a) \to \mathsf{Rew}\ r = \mathsf{Rew}\ s$

$\mathsf{Delta} : \Pi\mathsf{Rew}\ (\lambda x{:}o.\ \mathsf{Pair}\ (\mathsf{App}\ y^o\, x)\, (\mathsf{App}\ z^o\, x)).\ \Pi a{:}o.$
$\mathsf{App}\ (\mathsf{Rew}\ (\lambda x{:}o.\ \mathsf{Pair}\ (\mathsf{App}\ y\, x)\, (\mathsf{App}\ z\, x)))\, a =$
$\mathsf{Rew}\ (\lambda v{:}o.\ \mathsf{Pair}\ (\mathsf{App}\ (\mathsf{App}\ y\, a)\, v)\, (\mathsf{App}\ (\mathsf{App}\ z\, a)\, v))$

Figure 2. Classical RHO Encoding using PLF as a Metalanguage

3 Examples

3.1 PLF *as a metalanguage*

Readers familiar with the activity of encoding systems in LF will surely appreciate the usability of this metalanguage and will play with it providing only one, crucial example, namely the encoding of the untyped *alter ego* of PLF *i.e.* the RHO in PLF. This example illustrates how patterns can safely

relace sub-categories and coercions in the encoding of syntax. The Rewriting calculus [4,5], is a simple higher-order calculus where functions can be applied only upon reception of an argument whose "pattern" matches with the one explicitly declared in the function itself. This allows to represents naturally classical lambda calculus and many term rewriting systems. What makes the rewriting calculus appealing for reasoning on the web is precisely its foundational features that allow us to represent the atomic actions (*i.e.* rules) and the chaining of these actions (*i.e.* strategies) in order to achieve a global goal like, for example, transforming semi-structured data, extracting informations or inferring new ones. As the matching mechanism of the calculus can be parameterized by a suitable matching theory, this allows us for example to express in a precise way how the semi-structured data should be matched. The encoding in PLF is shown in Figure 2. We omit adequacy results.

We conclude by recalling that the sophisticated shapes of patterns and the built-ins pattern matching facilities make PLF (and its extensions) suitable for modeling regular languages, theory of functions, and term rewriting systems that are the engine of many HXML manipulations.

3.2 PLF *as a rule based language*

We present some HXML-oriented routines, written in the untyped PLF, *i.e.* the RHO [14], that can be either run and certified in our PLF interpreter [14]. In Figure 3, the first example find in an HXML catalogue the n-th item, the second example selects all the Italians items, the third example translates all the Italians items in the catalogue into French ones, and the last example drop all the Italians items.

3.3 During the workshop

During the workshop (when showing the encoding of the Rewriting calculus in PLF), Shriram Krishnamurthi raised an interesting question about the strong normalization of the type system of PLF. Since PLF is a direct derivate of the rewriting calculus, and since the latter has another type system that does not normalize (see *e.g.* [5]), the question was pertinent. The strong normalization result of the PLF type system is showed in a companion technical paper [10]. The encoding was about the dynamic semantics of the untyped Rewriting calculus. We can encode also one of the many different type systems for the Rewriting calculus, *e.g.* the one presented in [14].

Another interesting question, raised by Anthony Finkelstein, was about the need of fixpoints and recursive functions in web verification routines. The answer was suggested by Jan Scheffczyk and myself: very often we would need to verify some properties in a web page *and recursively* in all pages pointed by the page itself via `href` (like in the example presented in the Introduction); in this case, the presence of recursion via a fixpoint may be definitively useful.

```
*** Some list of numbers  ***
ZERO  = 0;;              ONE  = (succ 0);;       TWO = (succ ONE);;   THREE = (succ TWO);;
FOUR  = (succ THREE);;   FIVE = (succ FOUR);;    SIX = (succ FIVE);;  SEVEN = (succ SIX);;
EIGHT = (succ SEVEN);;   NINE = (succ EIGHT);;   TEN = (succ NINE);;

*** Some list of friends ***
ME   = (person ((first luigi)   ,(last liquori) ,(sex m),(empl inria)   ,(nat it) ,(cat ONE)));;
YOU  = (person ((first jessica) ,(last rabbit)  ,(sex f),(empl disney)  ,(nat usa),(cat TWO)));;
SHE  = (person ((first helene)  ,(last kirchner),(sex f),(empl cnrs)    ,(nat fr) ,(cat ZERO)));;
HIM  = (person ((first claude)  ,(last kirchner),(sex m),(empl inria)   ,(nat fr) ,(cat FOUR)));;
HER  = (person ((first uma)     ,(last thurman) ,(sex f),(empl hollywd) ,(nat usa),(cat FIVE)));;
BIG  = (person ((first bg)      ,(last sidharth),(sex m),(empl birla)   ,(nat in) ,(cat SIX)));;
HEAD = (person ((first moreno)  ,(last falaschi),(sex m),(empl siena)   ,(nat it) ,(cat TWO)));;
BOSS = (person ((first furio)   ,(last honsell) ,(sex m),(empl udine)   ,(nat it) ,(cat ONE)));;
JEFE = (person ((first maria)   ,(last alpuente),(sex f),(empl valencia),(nat es) ,(cat ZERO)));;
GURU = (person ((first salvador),(last lucas)   ,(sex m),(empl papaya)  ,(nat es) ,(cat ONE)));;

*** The DB ***
DB = (group (ME,YOU,SHE,HIM,HER,BIG,HEAD,BOSS,JEFE,GURU,nil));;

*** FINDN: Find in a DB the nth Element in a xml catalogue ***
[FINDN = ((0,nil)                -> fail, ((succ N),(group nil))   -> fail,
         ((succ 0),(group (X,Y))) -> X,    ((succ N),(group (X,Y))) -> (FINDN (N,(group Y))))];
(FINDN (THREE,DB));;

*** SELECTIT: Select in a DB the all the items of "it" nationality  ***
[SELECTIT  =
           ((group (nil))                          -> (nil),
            (group ((person (X, (nat it) ,V)),Z)) -> ((person (X,(nat it) ,V)),
                                                      (SELECTIT (group Z))),
            (group ((person (X,\(nat it) ,V)),Z)) -> (SELECTIT (group Z)))];;
(SELECTIT DB);;

*** NOITWFR Translate all the ``it'' items into ``fr'' items ***
[NOITWFR = ((group (nil))                       -> (nil),
           (group ((person (X,(nat  it),V)),Z)) -> (NOITWFR (group Z)),
           (group ((person (X,(nat \it),V)),Z)) -> (group ((person (X,(nat fr),V)),
                                                   (NOITWFR (group Z)))))];;
(NOITWFR DB);;

*** DROPIT: Kill in a DB the all the items of "it" nationality  ***
[DROPIT  = ((group (nil))                        -> (nil),
           (group ((person (X,  (nat it),V)),Z)) -> (DROPIT (group Z)),
           (group ((person (X,Y\(nat it),V)),Z)) -> ((person (X,Y,V)),(DROPIT (group Z))))];;
(DROPIT DB);;
```

Figure 3. Four examples using the Untyped PLF(*i.e.* the RHO) as Rule-based Language

Acknowledgments

The authors are also sincerely grateful to Benjamin Wack for point out a bug in the encoding of Delta, and all the anonymous referees for the suggestions they give to improve the final presentation. This work is supported by grant Aeolus FP6-2004-IST-FET Proactive, and the French grant ACI Modulogic.

References

[1] M. Alpuente, D. Ballis, and M. Falaschi. A Rewriting-based Framework for Web Sites Verification. In *Proc. of RULE*, volume 124/1 of *ENTCS*, pages 41–61, 2005.

[2] D. Ballis and J. Garcia-Vivo. A Rewriting-based system for Web site

Verification. In *Proc. of WWV*, pages 153–156, 2005.

[3] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Pattern Type Systems. In *Proc. of POPL*. The ACM Press, 2003.

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

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

[6] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[7] D. J. Dougherty. Adding Algebraic Rewriting to the Untyped Lambda Calculus. *Information and Computation*, 101(2):251–267, 1992.

[8] A. Finkelstein. Business Data Validation: lessons from practice. In *Proc. of WWV*, page 1, 2005.

[9] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, 1992.

[10] F. Honsell and L. Liquori. The Pattern Logical Framework. Manuscript.

[11] J.P. Jouannaud and M. Okada. Executable Higher-Order Algebraic Specification Languages. In *Proc. of LICS*, pages 350–361, 1991.

[12] C. Kirchner, H. Kirchner, and A. Santana. Anchoring modularity in HTML. In *Proc. of WWV*, 2005.

[13] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[14] L. Liquori and B. Serpette. iRho, An Imperative Rewriting Calculus. In *Proc. of PPDP*, pages 167–178. The ACM Press, 2004. http://www-sop.inria.fr/mirho/Luigi.Liquori/iRho/.

[15] S. Lucas. Rewriting-based navigation of Web sites. In *Proc. of WWV*, 2005.

[16] M. Okada. Strong Normalizability for the Combined System of the Typed λ Calculus and an Arbitrary Convergent Term Rewrite System. In *Proc. of ISSAC*, pages 357–363. ACM Press, 1989.

[17] Rho Team. The Rho Home Page, 2005. http://rho.loria.fr/.

[18] V. van Oostrom. Lambda Calculus with Patterns. TR IR-228, Vrije Univ. Amsterdam, 1990.