Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 172 (2007) 399–436
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Framework for Defining Logical Frameworks*

Furio Honsell[1]

DIMI, University of Udine, Italy

Marina Lenisa[2]

DIMI, University of Udine, Italy

Luigi Liquori[3]

INRIA, Sophia Antipolis, France

Abstract

In this paper, we introduce a *General Logical Framework*, called GLF, for defining Logical Frameworks, based on dependent types, in the style of the well known Edinburgh Logical Framework LF. The framework GLF features a generalized form of lambda abstraction where β-reductions fire provided the argument satisfies a logical predicate and may produce an n-ary substitution. The type system *keeps* track of when reductions have yet to fire. The framework GLF subsumes, by simple instantiation, LF as well as a large class of generalized constrained-based lambda calculi, ranging from well known restricted lambda calculi, such as Plotkin's call-by-value lambda calculus, to lambda calculi with patterns. But it suggests also a wide spectrum of new calculi which have intriguing potential as Logical Frameworks.
We investigate the metatheoretical properties of the calculus underpinning GLF and illustrate its expressive power. In particular, we focus on two interesting instantiations of GLF. The first is the Pattern Logical Framework (PLF), where applications fire via *pattern-matching* in the style of Cirstea, Kirchner, and Liquori. The second is the Closed Logical Framework (CLF) which features, besides standard β-reduction, also a reduction which fires only if the argument is a *closed* term. For both these instantiations of GLF we discuss standard metaproperties, such as subject reduction, confluence and strong normalization.
The GLF framework is particularly suitable, as a metalanguage, for encoding rewriting logics and logical systems, where rules require proof terms to have special syntactic constraints, *e.g.* logics with *rules of proof*, in addition to *rules of derivations*, such as, *e.g.*, modal logic, and call-by-value lambda calculus.

Keywords: Logical Framework, Lambda calculus, Pattern matching, Dependent-type systems, Logics

Dedicated to Gordon D. Plotkin, on the occasion of his 60^{th} *birthday*

* Work partially supported by UE Projects IST-CA-510996 TYPES, IST-015964 AEOLUS, and by Italian MIUR Project PRIN 2005015824 ART.

[1] Email: honsell@dimi.uniud.it

[2] Email: lenisa@dimi.uniud.it

[3] Email: Luigi.Liquori@inria.fr

doi:10.1016/j.entcs.2007.02.014

1 Introduction

Although LF, very rightly so, allows to encode rules as functions from proofs to proofs, it is nevertheless a little restrictive as to the "side conditions" that it can enforce on the application of rules. Rule application being encoded simply as lambda application, there are only roundabout ways to encode provisos, even as simple as that appearing in a *rule of proof*. Recall that a rule of proof can be applied only to premises which do not depend on any assumption, as opposed to a *rule of derivation* which can be applied everywhere. Also rules which appear in many natural deduction presentations of modal and program logics are very problematic in standard LF. Many such systems feature rules which can be applied only to premises which depend solely on assumptions of a particular shape [6], or whose derivation has been carried out using only certain sequences of rules. Finally, Linear or Relevance Logics appear to be encodable only using a very heavy machinery.

In the past, extensions of LF have often been proposed. The price to pay, however, was always very high as far as the language theory. The *desideratum* has always been that of having a metalogical framework, *i.e.* a *telescope* of systems, each a conservative extension of the previous ones, which can incrementally and naturally encode nastier and nastier classes of side-conditions. This is precisely what we propose in this paper.

The key idea is extremely simple. It amounts to removing a *blind spot*, thus making explicit two different notions, which are conflated to only one, in the original LF, *i.e.* which are taken to be definitionally equal. As already mentioned much of the rigidity of LF arises from the fact that β-reduction can be applied too generally. One would like to restrict it, but the type system appears not to be rich enough to be able to express such restrictions. What we propose is to use as type of an application, in the term application rule, (O·Appl) below, not the type which is obtained by carrying out directly in the metalanguage the substitution of the argument in the type, but a new form of type which simply records the information that such a reduction needs to be carried out. An application of the Type Conversion Rule can then recover the usual effect of the application rule. The old rule and the new rule (O·Appl′) appear as follows.

$$\frac{\Gamma \vdash M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau[N/x]}\ (O{\cdot}Appl) \qquad \frac{\Gamma \vdash M : \Pi x{:}\sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : (\lambda x{:}\sigma.\tau)\,N}\ (O{\cdot}Appl')$$

As it is often said: sometimes, less is more. And once this move has been made, we have a means of annotating in a type the information that a reduction is waiting to be carried out in the term. If we take seriously this move, such a type need not be necessarily *definitionally equal* to the reduced one as in the case of LF and we can generalize further our approach. Without much hassle, in effect, we have a principled and natural way of typing generalized calculi featuring generalized or restricted forms of β-reduction which wait for some constraint to be satisfied before they can fire. Each such calculus can be considered as a potential candidate for underpinning a new Logical Framework, where all the extra complexity in terms can be naturally tamed utilizing the expressive power of the new typing system. Once

this program is carried out in a sufficiently modular form, we have the telescopic metalogical framework we were looking for.

In order to proceed in full generality we introduce a new form of λ and corresponding Π abstraction $\Pi\mathcal{P}{:}\Delta.\tau$ and $\lambda\mathcal{P}{:}\Delta.M$. The unary predicate $\mathcal{P}$ is completely general at this stage, and the type context $\Delta \triangleq [x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n]$ denotes the variables bound by Π and λ in $\Pi\mathcal{P}{:}\Delta.\tau$ and $\lambda\mathcal{P}{:}\Delta.M$. We will show in the paper that it can be instantiated in various useful ways. For instance, it can enforce the fact that the argument is closed, or that all its free variables have a type of a given form. This format can also recover many existing calculi in the literature such as LF, the Rewriting Calculus [9,10], and the Plotkin's call-by-value lambda calculus [35]. In all cases, an application of the "type equality" rule can be used to recover, *conservatively*, the effect of successful β-reductions:

$$(\lambda\mathcal{P}{:}\Delta.M)\,N \longrightarrow M\widehat{\mathcal{P}}(N) \quad \text{provided } \mathcal{P}(N) \text{ holds and } \widehat{\mathcal{P}}(N) \text{ is a substitution.}$$

The extra types deriving from failures allow for precisely the extra elbow-room that is needed to prevent the applications of certain rules too loosely. It is now immediate to see that rules of proof can be dealt with straightforwardly by restricting applications to closed terms.

This idea of distinguishing between two notions which were previously flattened into one is a small step for a type system but a momentous step for a Logical Framework. The idea of capitalizing on the similarities between the "λ" and "Π" operators is not new, see *e.g.* [14, 23], but what we do here is to capitalize on it, in the type system, as was done in the work by Cirstea, Kirchner, and Liquori *The Rho Cube* [10]. By so doing, we allow for a generalized form of pattern lambda calculi, and also go beyond.

The papers which are most influential for our proposal and which we are most indebted with are [10] and [4]. The former is the paper which first puts to use the decomposition of the rule (O·Appl′) in special cases. It presents a collection of type systems for a typed variant of the Rewriting Calculus [4], which was later generalized in [4] to Pure Type Systems with patterns.

Summing up, we propose a General Logical Framework GLF and the General Lambda Calculus GL underpinning it. GLF, in that it accomodates various strong definitional equalities, can be viewed as a logical framework in the spirit of the, so called, *Poincaré principle*, or the more recent *Deduction Modulo* of [16]. The idea behind these approaches can be put briefly as follows. It is well-known that LF behaves like first-order logic. One can always encode explicitly, *i.e.* axiomatically, whatever judgment is necessary. However, both theoretically, *e.g.* in Martin-Löf systems, as well as pragmatically, *e.g.* in Coq, it is very useful to have stronger notions of definitional equality than pure β-equality. In the most recent approaches to formal proofs, one has subtle interplays between deduction and computation of definitional equality. In all these cases, however, each new definitional equality has to be justified outside the framework.

[4] This version of the Rewriting Calculus was a kind of typed lambda calculus with constants, algebraic patterns, and built-in matching constructions.

In this paper, we provide general results concerning classes of calculi which provide useful definitional equalities. In particular, we carry out an extensive investigation of the language theory of two important instantiations of GLF, called PLF and CLF respectively. The first features a general form of pattern β-reduction, while the second subsumes LF but it provides also a form of β-reduction restricted only to closed arguments.

1.1 Historical Remarks

A short recollection, by the first author, from exactly twenty years ago.

The Edinburgh LF took a rather short time to blossom: essentially the spring of 1986. A General Interactive Proof Development Environment was one of the first three projects of, what was then, the recently established Laboratory for the Foundations of Computer Science, LFCS, in Edinburgh. According to its first director, Robin Milner, the Laboratory was supposed to develop theoretically principled applications, in the spirit that Computer Science is also an experimental science. The goal of this project was a general interactive proof assistant which could provide a large number of proof editing, proof checking, and proof searching facilities for an arbitrary logical system as were available, at the time, in tools such as LCF [33] or NuPrl [11], only for specific formal systems. The challenge was that of not having to duplicate the implementation effort each time an interactive environment for a new logic was needed. The idea was that of developing a general theory of logical systems, which factored out uniformities across a wide class of logics and then of implementing, once and for all, a general logic-independent proof development environment based on such a theory. This general environment could then be tailored to a specific system, without having to re-implement everything from scratch each time.

In the early months of 1986 Gordon Plotkin started experimenting with typed lambda calculi, supporting the *proposition-as-types paradigm*, as a general metalanguage and framework for logical systems. A few researchers at LFCS joined in, and by midsummer 1986 the *Framework for Defining Logics* [19] as it was presented to the LICS conference in 1987, was pretty much finalized.

It was immediately clear that the higher order nature of the Dependent Typed Lambda Calculus, later to be known as LF, was particularly satisfactory as a general metalanguage for expressing logical languages, binding operators, rules, and proof development. What appeared in the traditional presentations of logical systems as intricate idiosyncrasies and strange provisos in rules, either completely disappeared in the LF encoding of the system or were greatly clarified. An encoding of a logic in the Framework always turned out to be particularly insightful in understanding the system itself, to the point that LF appeared as normative. The conclusion was that LF was the most suitable type system introduced so far to play the role of a metalanguage for logics presented in natural deduction style. It was the perfect medium to implement the newly formulated *judgments-as-types* paradigm. Furthermore, LF subsumed also a number of previous ideas in formal mathematics and proof theory stemming from the Automath tradition [14, 29], Constructive Type Theory [27, 7]

and it capitalized on the notion of *judgment* as discussed by Martin-Löf in a series of papers in the mid '80's, [28].

The Logical Framework game, triggered by LF, became rapidly quite popular in the formal proof development community and many authors [18,8] played it on their systems. Since then, Logical Frameworks, logical metalanguages, and general proof assistants grew up to a well defined, and very active sector of Logic and Computer Science. It benefited considerably by the results stemming from the community working on Constructive Type Theory as a framework for formalizing mathematics, [3, 12]. Nowadays there are a number of specific conferences that address these topics, *e.g. Merλin, Theorem Proving in Higher Order Logic, Logical Framework Metalanguages: theory and practice*, a vast literature, see *e.g.* [26, 36, 5, 34] and an almost twenty years old EU Working Group community, called Types, actively working on Type Theory [38].

Since the birth of LF, the challenge was that of assessing the expressive power of the metalanguage, or equivalently that of coming up with logics which could break the Framework. LF proved to be particularly successful in dealing with metavariables, variable scoping and binding, Higher Order Abstract Syntax and, with a little effort, also with names [15,20], program and modal logics [2,1].

Synopsis.

The paper is structured as follows. In Section 2, we present the syntax of GL and the type system of GLF. We discuss general properties of GLF and present several instantiations of GLF to known as well as new calculi. In Section 3, we discuss an important instantiation of GLF, the Pattern Logical Framework, called PLF, where reductions fire via pattern-matching. A thorough investigation of the metatheoretical properties of PLF is carried out. In Section 4, we present another instantiation of the GLF framework, CLF which features besides standard β-reduction also a β-reduction restricted to closed terms. In Section 5, we illustrate the expressive power of these new typed calculi as metalanguages. In particular we give a shorter, and possibly sharper, encoding of Plotkin's call-by-value lambda calculus in PLF capitalizing on algebraic patterns, and an encoding in CLF of rules of proof in *Modal Logics*. Conclusions and directions for future work appear in Section 6. Proofs of main theorems appear in the Appendix.

2 The General Logical Framework

In this section, we present the General Lambda Calculus GL and we discuss the language theory underpinning the General Logical Framework GLF.

General Notations.

Let $M, N, \ldots \in \mathcal{O}$ denote terms (a.k.a. objects), $\sigma, \tau, \ldots \in \mathcal{F}$ denote types (a.k.a. families), $a, b, c, \ldots$ denote constant types, $K \in \mathcal{K}$ denote kinds, $x, y, z, \ldots$ denote variables, $f, g, \ldots$ denote term constants, $\Gamma, \Delta \in \mathcal{C}$ denote contexts, $\Sigma \in \mathcal{S}$ denote signatures, and let $\mathcal{P}, \mathcal{Q}, \ldots$ range over a set of logical predicates $\mathcal{L}$. All symbols can

appear indexed. The symbol $\equiv$ denotes syntactic identity on terms. Terms will be taken up to α-conversion.

2.1 The General Typed Lambda Calculus

The General Typed Lambda Calculus, called GL, is a generalization of the typed lambda calculus *à la* Church with constants, but it allows unary logical predicates instead of simple variables in lambda abstractions. The syntax of GL terms is given below, type families will be defined later.

Definition 2.1 (GL Terms a.k.a. Objects)

$$\begin{array}{lll} M, N \in \mathsf{GL} & M ::= f \mid x \mid \lambda\mathcal{P}{:}\Delta.M \mid M\,N & \textit{Terms} \\ \Gamma, \Delta \in \mathcal{C} & \Gamma ::= \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts} \\ \sigma, \tau \in \mathcal{F} & \sigma ::= \ldots & \textit{Types} \\ \mathcal{P}, \mathcal{Q} \in \mathcal{L} & \mathcal{P} ::= \ldots & \textit{(unary) Predicates} \end{array}$$

where the variables in $\mathsf{Dom}(\Delta)$ *occurring in* M *are bound in* $\lambda\mathcal{P}{:}\Delta.M$.

The term $\lambda\mathcal{P}{:}\Delta.M$ is called a *predicate abstraction*. The intuition behind a generalized β-redex of the shape $(\lambda\mathcal{P}{:}\Delta.M)\,N$ is that the argument N of the function can be propagated in the body M, and the redex progresses to $M\theta$, for a suitable substitution θ over $\mathsf{Dom}(\Delta)$, provided the unary predicate $\mathcal{P}$ holds on N. Otherwise the term *is stuck*. The language GL is parametrized over a set of unary predicates $\mathcal{P}$, which we do not specify.

Definition 2.2 (Auxiliary Functions) (i) *Let* $\bar{}: [\mathcal{L} \to [\mathcal{C} \to \mathsf{GL}]]$ *be a function taking a predicate* $\mathcal{P}$ *and a context* Δ*, and producing a term whose free variables are exactly those in* $\mathsf{Dom}(\Delta)$. *We denote* $\overline{\mathcal{P}}(\Delta)$ *simply by* $\overline{\mathcal{P}}$.

(ii) *Let* $\widehat{\ }: [\mathcal{L} \to [\mathcal{C} \to [\mathsf{GL} \to \mathcal{S}ub]_\perp]]$ *be a function taking a* $\mathcal{P}$ *and a* Δ*, and producing a partial function that takes a term* M *and produces a substitution over* $\mathsf{Dom}(\Delta)$*, provided* M *satisfies* $\mathcal{P}$. *Informally,* $\mathcal{P}$ *is a logical filter that constrains reductions. We denote* $\widehat{\mathcal{P}}(\Delta)$ *simply by* $\widehat{\mathcal{P}}$.

The next definition introduces the standard notions of top-level, one-step, many-steps β-reduction, and its congruence closure.

Definition 2.3 (One-step/Many-Steps Reduction, Congruence) (i) *For every predicate* $\mathcal{P} \in \mathcal{L}$*, the top-level reduction is defined as*

$$(\beta_{\mathcal{P}})\ (\lambda\mathcal{P}{:}\Delta.M)\,N \to_{\beta_{\mathcal{P}}} M\,\widehat{\mathcal{P}}(N) \qquad \text{if } \mathcal{P}(N) \text{ holds}$$

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

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

Let $\mathsf{C}[M]$ be the result of filling the hole with the term M. The one-step evaluation $\mapsto_{\beta_{\mathcal{P}}}$ is defined by the inference rule

$$\frac{M \to_{\beta_{\mathcal{P}}} N}{\mathsf{C}[M] \mapsto_{\beta_{\mathcal{P}}} \mathsf{C}[N]}\ (Ctx)$$

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

We use $\mapsto_\beta$ to denote $\bigcup_{\mathcal{P}\in\mathcal{L}} \mapsto_{\beta_{\mathcal{P}}}$, similarly $\mapsto\!\!\!\!\to_\beta$ and $=_\beta$ will denote the unions of all $\mapsto\!\!\!\!\to_{\beta_{\mathcal{P}}}$'s and all $=_{\beta_{\mathcal{P}}}$'s, respectively.

2.2 The General Logical Framework

The General Logical Framework, called GLF, is a dependent type system for the General Typed Lambda Calculus GL. In a nutshell, there are two main generalizations with respect to a standard dependent type theory *à la* LF:

(i) The LF product-type $\Pi x{:}\sigma.\tau$ is replaced in GLF by the more general constrained product-type $\Pi\mathcal{P}{:}\Delta.\tau$ that will be inhabited by a predicate-abstraction of the shape $\lambda\mathcal{P}{:}\Delta.M$.

(ii) In the typing rule for application one usually has that the final type for $M\,N$ is $\tau[N/x]$ where the notation $[N/x]$ means the meta-operation of substituting every occurrence of x with the object term N. In GLF, this meta notation for the type of the application is taken seriously and is represented by a GLF dependent-type not necessarily in normal form $(\lambda\mathcal{P}{:}\Delta.\tau)\,N$. This term reduces to the dependent-type $\tau\widehat{\mathcal{P}}(N)$ if $\mathcal{P}(N)$ holds (and $\widehat{\mathcal{P}}(N)$ is a substitution), otherwise it gets stuck. Of course, if the reduction fires, via a standard type conversion rule, the reduced type is inhabited by the application $M\,N$.

2.2.1 Syntax.

The syntax of GLF families is defined as follows.

Definition 2.4 (GLF Types a.k.a. Families)

$$\sigma,\tau \in \mathcal{F} \qquad \sigma ::= a \mid \Pi\mathcal{P}{:}\Delta.\sigma \mid \lambda\mathcal{P}{:}\Delta.\sigma \mid \sigma\,M \qquad \textit{Types}$$

In the syntax, a is a constant type, or more generally, a curried type valued function, $\Pi\mathcal{P}{:}\Delta.\tau$ is a constrained product-type, $\lambda\mathcal{P}{:}\Delta.\tau$ is a constructor for type families, and $\sigma\,M$ as usual, is the type family produced by applying a type family of higher kind to a term.

To complete the presentation of GLF we need, as usual, suitable syntax for *signatures*, *contexts*, and *kinds* as follows.

$$
\begin{array}{rlll}
\Sigma \in \mathcal{S} & \Sigma & ::= & \emptyset \mid \Sigma, a{:}K \mid \Sigma, f{:}\sigma & \textit{Signatures} \\
\Gamma, \Delta \in \mathcal{C} & \Gamma & ::= & \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts} \\
K \in \mathcal{K} & K & ::= & \mathsf{Type} \mid \Pi\mathcal{P}{:}\Delta.K \mid \lambda\mathcal{P}{:}\Delta.K \mid K\,M & \textit{Kinds} \\
\sigma, \tau \in \mathcal{F} & \sigma & ::= & a \mid \Pi\mathcal{P}{:}\Delta.\sigma \mid \lambda\mathcal{P}{:}\Delta.\sigma \mid \sigma\,M & \textit{Types (Families)} \\
M, N \in \mathcal{O} & M & ::= & f \mid x \mid \lambda\mathcal{P}{:}\Delta.M \mid M\,N & \textit{Terms (Objects)}
\end{array}
$$

$$
\begin{array}{ll}
(\beta_{\mathcal{P}}\text{–Terms}) & (\lambda\mathcal{P}{:}\Delta.M)\,N \rightarrow_{\beta_{\mathcal{P}}} M\,\widehat{\mathcal{P}}(N) \\
(\beta_{\mathcal{P}}\text{–Types}) & (\lambda\mathcal{P}{:}\Delta.\tau)\,N \rightarrow_{\beta_{\mathcal{P}}} \tau\,\widehat{\mathcal{P}}(N) \\
(\beta_{\mathcal{P}}\text{–Kinds}) & (\lambda\mathcal{P}{:}\Delta.K)\,N \rightarrow_{\beta_{\mathcal{P}}} K\,\widehat{\mathcal{P}}(N)
\end{array}
$$

Figure 1. GLF Syntax and Operational Semantics

Definition 2.5 (GLF Signatures, Contexts and Kinds)

$$
\begin{array}{rlll}
\Sigma \in \mathcal{S} & \Sigma & ::= \emptyset \mid \Sigma, a{:}K \mid \Sigma, f{:}\sigma & \textit{Signatures} \\
\Gamma, \Delta \in \mathcal{C} & \Gamma & ::= \emptyset \mid \Gamma, x{:}\sigma & \textit{Contexts} \\
K \in \mathcal{K} & K & ::= \mathsf{Type} \mid \Pi\mathcal{P}{:}\Delta.K \mid \lambda\mathcal{P}{:}\Delta.K \mid K\,M & \textit{Kinds}
\end{array}
$$

In GLF, we introduce a reduction on kinds not in normal form $(\lambda\mathcal{P}{:}\Delta.K)\,M$ that, again, reduces to $K\widehat{\mathcal{P}}(M)$ if and only if $\mathcal{P}(M)$ is satisfied. Figure 1 summarizes the syntax and the operational semantics of GLF.

2.2.2 Type System.
As usual, the type system for GLF proves judgments of the shape:

$$\Sigma\ \mathsf{sig} \qquad \vdash_{\Sigma} \Gamma \qquad \Gamma \vdash_{\Sigma} K \qquad \Gamma \vdash \sigma : K \qquad \Gamma \vdash_{\Sigma} M : \sigma$$

The type system rules for GLF are presented in Figure 2. Notice that rule schemas $(*{\cdot}\mathsf{Pi}), (*{\cdot}\mathsf{Abs})$, and $(*{\cdot}\mathsf{Appl})$ are parametrized over the predicate $\mathcal{P}$. The inference rules make use of a notion of definitional equality, consisting of the following three forms of auxiliary judgments:

$$
\begin{array}{ll}
\Gamma \vdash_{\Sigma} K =_{\beta} K' & K \text{ and } K' \text{ are definitionally equal kinds in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash_{\Sigma} \sigma =_{\beta} \tau & \sigma \text{ and } \tau \text{ are definitionally equal types in } \Gamma \text{ and } \Sigma \\
\Gamma \vdash_{\Sigma} M =_{\beta} N & M \text{ and } N \text{ are definitionally equal terms in } \Gamma \text{ and } \Sigma
\end{array}
$$

The first two of these relations are used directly; the third one is used to define the others. We do not give the list of rules for these three judgments. These are

Signature rules

$$\frac{}{\emptyset\ \mathsf{sig}}\ (S{\cdot}Empty)$$

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K\ \mathsf{sig}}\ (S{\cdot}Kind)$$

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma \sigma : \mathsf{Type}\ f \notin \mathsf{Dom}(\Sigma)}{\Sigma, f{:}\sigma\ \mathsf{sig}}\ (S{\cdot}Type)$$

Context rules

$$\frac{\Sigma\ \mathsf{sig}}{\vdash_\Sigma \emptyset}\ (C{\cdot}Empty)$$

$$\frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma \sigma : \mathsf{Type}\ x \notin \mathsf{Dom}(\Gamma)}{\vdash_\Sigma \Gamma, x{:}\sigma}\ (C{\cdot}Type)$$

Kind rules

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

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi\mathcal{P}{:}\Delta.K}\ (K{\cdot}Pi)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.K}\ (K{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma \Pi\mathcal{P}{:}\Delta.K \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma (\lambda\mathcal{P}{:}\Delta.K)\, N}\ (K{\cdot}Appl)$$

Family rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Sigma}{\Gamma \vdash_\Sigma a : K}\ (F{\cdot}Var)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma \tau : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi\mathcal{P}{:}\Delta.\tau : \mathsf{Type}}\ (F{\cdot}Pi)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma \tau : K}{\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.\tau : \Pi\mathcal{P}{:}\Delta.K}\ (F{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : \Pi\mathcal{P}{:}\Delta.K \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \tau \qquad \Gamma \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \sigma\, M : (\lambda\mathcal{P}{:}\Delta.K)\, M}\ (F{\cdot}Appl)$$

$$\frac{\Gamma \vdash_\Sigma \sigma : K' \quad \Gamma \vdash_\Sigma K \qquad \Gamma \vdash_\Sigma K =_\beta K'}{\Gamma \vdash_\Sigma \sigma : K}\ (F{\cdot}Conv)$$

Object rules

$$\frac{\vdash_\Sigma \Gamma \quad f{:}\sigma \in \Sigma}{\Gamma \vdash_\Sigma f : \sigma}\ (O{\cdot}Const)$$

$$\frac{\vdash_\Sigma \Gamma \quad x{:}\sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}\ (O{\cdot}Var)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma, \Delta \vdash_\Sigma M : \tau}{\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.M : \Pi\mathcal{P}{:}\Delta.\tau}\ (O{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi\mathcal{P}{:}\Delta.\tau \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \qquad \Gamma \vdash_\Sigma N : \sigma}{\Gamma \vdash_\Sigma M\, N : (\lambda\mathcal{P}{:}\Delta.\tau)\, N}\ (O{\cdot}Appl)$$

$$\frac{\Gamma \vdash_\Sigma M : \sigma \quad \Gamma \vdash_\Sigma \tau : \mathsf{Type}\ \Gamma \vdash_\Sigma \sigma =_\beta \tau}{\Gamma \vdash_\Sigma M : \tau}\ (O{\cdot}Conv)$$

Figure 2. The GLF Type System

standard but for the fact that we have to consider multiple substitutions. By way of example we give only the main rule (Type·Eq) for type equality:

$$\frac{\forall y_i \in \mathsf{Dom}(\widehat{\mathcal{P}}(M)).\ [\ \Gamma, \Delta \vdash_\Sigma \widehat{\mathcal{P}}(M)(y_i) : \Delta(y_i)\ ] \quad \Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma \quad \Gamma \vdash_\Sigma M : \sigma}{\Gamma \vdash_\Sigma (\lambda\mathcal{P}{:}\Delta.\tau)\, M =_\beta \tau\, \widehat{\mathcal{P}}(M)}$$

2.3 *Instantiating* GL/GLF

The behavior of GL and GLF strongly depend on the precise nature of the predicates involved in abstractions. In general we can instantiate them as follows.

Definition 2.6 (General Predicate Set $\mathbb{S}$) *A General Predicate Set is* $\mathbb{S} \triangleq \{ (\mathcal{P}_i , \Delta_i , \overline{\mathcal{P}_i} , \widehat{\mathcal{P}_i}) \}_{i\in I}$, *where* $^-$ *and* $\widehat{\ }$ *are the functions of Definition 2.2.*

Definition 2.7 (General Predicate $\mathsf{GL}_\mathbb{S}$/$\mathsf{GLF}_\mathbb{S}$) *For a given* $\mathbb{S}$, *a Predicate Lambda Calculus (General Predicate Logical Framework), called* $\mathsf{GL}_\mathbb{S}$ ($\mathsf{GLF}_\mathbb{S}$), *can be obtained by restricting (instantiating) the predicates to the ones declared in* $\mathbb{S}$.

A list of desired properties for $\mathsf{GLF}_\mathbb{S}$ follows. Let α be any judgment in $\mathsf{GLF}_\mathbb{S}$.

Desiderata 1 (Desired Properties of $\mathsf{GLF}_\mathbb{S}$)

Subderivation Property • *Any derivation of* $\Gamma \vdash_\Sigma \alpha$ *has subderivations of* Σ sig *and* $\vdash_\Sigma \Gamma$;
- *Any derivation of* $\Sigma, a{:}K$ sig *has subderivations of* Σ sig *and* $\vdash_\Sigma K$;
- *Any derivation of* $\Sigma, f{:}\sigma$ sig *has subderivations of* Σ sig *and* $\vdash_\Sigma \sigma : \mathsf{Type}$;
- *Any derivation of* $\vdash_\Sigma \Gamma, x{:}\sigma$ *has subderivations of* Σ sig *and* $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$;
- *Given a derivation of* $\Gamma \vdash_\Sigma \alpha$ *and any subterm occurring in the subject of the judgment, there exists a derivation of a smaller length of a judgment having that subterm as a subject;*
- *If* $\Gamma \vdash_\Sigma \sigma : K$, *then* $\Gamma \vdash_\Sigma K$;
- *If* $\Gamma \vdash_\Sigma M : \sigma$, *then* $\Gamma \vdash_\Sigma \sigma : \mathsf{Type}$.

Derivability of Weakening and Permutation *If* Γ *and* Δ *are valid contexts, and every declaration occurring in* Γ *also occurs in* Δ, *then* $\Gamma \vdash_\Sigma \alpha$ *implies* $\Delta \vdash_\Sigma \alpha$.

Unicity of Types and Kinds • *If* $\Gamma \vdash_\Sigma M : \sigma$ *and* $\Gamma \vdash_\Sigma M : \tau$, *then* $\Gamma \vdash \sigma{=_\beta}\tau$;
- *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* $\Gamma \vdash_\Sigma \sigma : K'$, *then* $\Gamma \vdash_\Sigma K{=_\beta}K'$.

Transitivity *If* $\Gamma, x{:}\sigma, \Delta \vdash_\Sigma \alpha$ *and* $\Gamma \vdash_\Sigma M : \sigma$, *then* $\Gamma, \Delta[M/x] \vdash_\Sigma \alpha[M/x]$.

Abstraction Typing • *If* $\Gamma \vdash_\Sigma K$ *and* Γ' *is such that* $\mathsf{Dom}(\Gamma) = \mathsf{Dom}(\Gamma')$, *and for all* $x \in \mathsf{Dom}(\Gamma)$, $\Gamma \vdash_\Sigma \Gamma(x){=_\beta}\Gamma'(x)$ *and* $\mathsf{Fv}(\Gamma(x)) \subseteq \mathsf{Fv}(\Gamma'(x))$, *then* $\Gamma' \vdash_\Sigma K$;
- *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* Γ' *is such that* $\mathsf{Dom}(\Gamma) = \mathsf{Dom}(\Gamma')$, *and for all* $x \in \mathsf{Dom}(\Gamma)$, $\Gamma \vdash_\Sigma \Gamma(x){=_\beta}\Gamma'(x)$ *and* $\mathsf{Fv}(\Gamma(x)) \subseteq \mathsf{Fv}(\Gamma'(x))$, *then* $\Gamma' \vdash_\Sigma \sigma : K$;
- *If* $\Gamma \vdash_\Sigma M : A$ *and* Γ' *is such that* $\mathsf{Dom}(\Gamma) = \mathsf{Dom}(\Gamma')$, *and for all* $x \in \mathsf{Dom}(\Gamma)$, $\Gamma \vdash_\Sigma \Gamma(x){=_\beta}\Gamma'(x)$ *and* $\mathsf{Fv}(\Gamma(x)) \subseteq \mathsf{Fv}(\Gamma'(x))$, *then* $\Gamma' \vdash_\Sigma M : A$;
- *If* $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.\tau : \Pi\mathcal{P}'{:}\Delta'.K$, *then* $\mathsf{Dom}(\Delta) = \mathsf{Dom}(\Delta')$, *and for all* $x \in \mathsf{Dom}(\Delta)$, *we have* $\Gamma, \Delta \vdash_\Sigma \Delta(x){=_\beta}\Delta'(x)$, *and* $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}}{=_\beta}\overline{\mathcal{P}}'$;
- *If* $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.M : \Pi\mathcal{P}'{:}\Delta'.\tau$, *then* $\mathsf{Dom}(\Delta) = \mathsf{Dom}(\Delta')$, *and for all* $x \in \mathsf{Dom}(\Delta)$, *we have* $\Gamma, \Delta \vdash_\Sigma \Delta(x){=_\beta}\Delta'(x)$, *and* $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}}{=_\beta}\overline{\mathcal{P}}'$;
- *If* $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.\tau : \Pi\mathcal{P}{:}\Delta.K$, *then* $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma$, *and* $\Gamma, \Delta \vdash_\Sigma \tau : K$;
- *If* $\Gamma \vdash_\Sigma \lambda\mathcal{P}{:}\Delta.M : \Pi\mathcal{P}{:}\Delta.\tau$, *then* $\Gamma, \Delta \vdash_\Sigma \overline{\mathcal{P}} : \sigma$, *and* $\Gamma, \Delta \vdash_\Sigma M : \tau$.

Subject Reduction • *If* $\Gamma \vdash_\Sigma K$ *and* $K \to_\beta K'$, *then* $\Gamma \vdash_\Sigma K'$;
- *If* $\Gamma \vdash_\Sigma \sigma : K$ *and* $\sigma \to_\beta \tau$, *then* $\Gamma \vdash_\Sigma \tau : K$;

- *If* $\Gamma \vdash_\Sigma M : \sigma$ *and* $M \to_\beta N$, *then* $\Gamma \vdash_\Sigma N : \sigma$.

Confluence • *If* $K_1 \mapsto\!\!\!\!\to_\beta K_2$ *and* $K_1 \mapsto\!\!\!\!\to_\beta K_3$, *then there exists* K_4 *such that* $K_2 \mapsto\!\!\!\!\to_\beta K_4$, *and* $K_3 \mapsto\!\!\!\!\to_\beta K_4$;
- *If* $\sigma_1 \mapsto\!\!\!\!\to_\beta \sigma_2$ *and* $\sigma_1 \mapsto\!\!\!\!\to_\beta \sigma_3$, *then there exists* σ_4 *such that* $\sigma_2 \mapsto\!\!\!\!\to_\beta \sigma_4$, *and* $\sigma_3 \mapsto\!\!\!\!\to_\beta \sigma_4$;
- *If* $M_1 \mapsto\!\!\!\!\to_\beta M_2$ *and* $M_1 \mapsto\!\!\!\!\to_\beta M_3$, *then there exists* M_4 *such that* $M_2 \mapsto\!\!\!\!\to_\beta M_4$, *and* $M_3 \mapsto\!\!\!\!\to_\beta M_4$.

Strong Normalization • *If* $\Gamma \vdash_\Sigma K$, *then* K *is strongly normalizing;*
- *If* $\Gamma \vdash_\Sigma \sigma : K$, *then* σ *is strongly normalizing;*
- *If* $\Gamma \vdash_\Sigma M : \sigma$, *then* M *is strongly normalizing.*

Judgments decidability *It is decidable whether* $\Gamma \vdash_\Sigma \alpha$ *is derivable.*

The following is about the most that one could prove for a General Logical Framework at this stage of generality.

Conjecture 2.8 (General Properties of GLF) *The following properties are valid in* GLF*:*

- *Subderivation;*
- *Derivability of Weakening and Permutation;*
- *Unicity of Types and Kinds;*
- *Abstraction Typing;*
- *Subject Reduction.*

2.4 Simple Examples

We illustrate the General Lambda Calculus and the General Logical Framework through some simple instantiations. More lambda calculi and logical frameworks can be captured by GLF, using appropriate general predicate sets $\mathbb{S}$'s.

2.4.1 The Typed Lambda Calculus à la Church.

The set $\mathbb{S}_{\text{Church}}$ is $\mathbb{S}_{\text{Church}} \triangleq \{ (\ \text{True}_x\ ,\ [x{:}\sigma]\ ,\ \overline{\text{True}_x}\ ,\ \widehat{\text{True}_x}\ )^{x\in\mathbb{V}} \}$, where True_x is $\text{True}_x(M) \triangleq \mathit{true}\ \ (\forall M)$, and $\overline{\text{True}_x} \triangleq x$, and $\widehat{\text{True}_x}(M) \triangleq [M/x]$. Notice that the freshness of the variable x is enforced in the typing rules by the well-formedness of contexts.

2.4.2 Plotkin's Call-by-Value Lambda Calculus.

The set $\mathbb{S}_{\beta_\mathsf{v}}$ is $\{ (\ \text{Value}_x\ ,\ [x{:}\sigma]\ ,\ \overline{\text{Value}_x}\ ,\ \widehat{\text{Value}_x}\ )^{x\in\mathbb{V}} \}$, where Value_x is

$$\text{Value}_x(M) \triangleq \begin{cases} \mathit{true} & \text{if } M \text{ is a variable or an abstraction} \\ \mathit{false} & \text{otherwise} \end{cases}$$

and $\overline{\text{Value}_x} \triangleq x$, and $\widehat{\text{Value}_x}(M) \triangleq$ if M is a variable or an abstraction then $[M/x]$, else $\bot$.

2.4.3 The Closed Typed Lambda Calculus.
The set $\mathbb{S}_\emptyset$ is $\{ (\mathrm{Closed}_x , [x{:}\sigma] , \overline{\mathrm{Closed}_x} , \widehat{\mathrm{Closed}_x})^{x\in\mathbb{V}} \}$, where Closed_x is

$$\mathrm{Closed}_x(M) \triangleq \begin{cases} \mathit{true} & \text{if } \mathsf{Fv}(M) = \emptyset \\ \mathit{false} & \text{otherwise} \end{cases}$$

and $\overline{\mathrm{Closed}_x} \triangleq x$, and $\widehat{\mathrm{Closed}_x}(M) \triangleq$ if $\mathsf{Fv}(M) = \emptyset$, then $[M/x]$, else $\bot$.

2.4.4 The Rewriting Calculus à la Cirstea-Kirchner-Liquori.
The set $\mathbb{S}_{\mathrm{Rho}}$ is: $\{ (\mathrm{Match}_{P_i} , \Delta_i , \overline{\mathrm{Match}_{P_i}} , \widehat{\mathrm{Match}_{P_i}})^{i\in I} \}$, where the predicate Match_{P_i} is defined as follows.

$$\mathrm{Match}_{P_i}(M) \triangleq \begin{cases} \mathit{true} & \text{if } \exists\theta_i.\ \mathcal{A}lg(P_i; M) = \theta_i \text{ and } \mathsf{Nf}(P_i) \\ \mathit{false} & \text{otherwise} \end{cases}$$

where

- the predicate $\mathsf{Nf}(P_i)$ is true if and only if P_i has a $\rightarrow_{\beta\delta}$-normal form,
- $\mathcal{A}lg$ is essentially the matching algorithm defined in [4] (where $\mathsf{Fv}(P_i) = \mathsf{Dom}(\Delta_i)$), which provides a substitution if M matches with the pattern P, and fails, otherwise, and
- $\overline{\mathrm{Match}_{P_i}} \triangleq P_i$, and $\widehat{\mathrm{Match}_{P_i}}(M) \triangleq$ if $\exists\theta_i.\ \mathcal{A}lg(P_i; M) = \theta_i$, then θ_i, else $\bot$.

This calculus is equivalent to the class of *functional* Pure Type Systems with Patterns of [4]. A specific version of GLF, which features rather general shapes of patterns, but nevertheless has a considerably rich theory of expressions, will be introduced and studied in Section 3.

2.4.5 The Edinburgh's Logical Framework à la Harper-Honsell-Plotkin.
The set $\mathbb{S}_{\mathrm{LF}}$ is $\mathbb{S}_{\mathrm{Church}}$. The function $[\![\ ]\!]$ is essentially a function that replaces every occurrence of True_x by x.

2.4.6 The Closed Logical Framework CLF.
The set $\mathbb{S}_{\mathrm{CLF}}$ is:
$\{ (\mathrm{True}_x , [x{:}\sigma] , \overline{\mathrm{True}_x} , \widehat{\mathrm{True}_x})^{x\in\mathbb{V}} , (\mathrm{Closed}_x , [x{:}\sigma] , \overline{\mathrm{Closed}_x} , \widehat{\mathrm{Closed}_x})^{x\in\mathbb{V}} \}$, where True_x and Closed_x are defined as before. The Closed Logical Framework CLF combines two notions of β-reduction, the standard β-reduction and the β-reduction restricted to closed arguments. This Logical Framework will be extensively studied in Section 4.

3 The Pattern Logical Framework

Since the introduction of the Logical Framework in [19], blending dependent typed lambda calculi with rewriting systems has been a major challenge, see [30,22,17,24, 32,9,4,13,39]. When the lambda calculus underpinning a logical framework features also rewriting rules, there is potential for enhancing the pragmatic usability of the system. More natural and transparent encodings can be provided (see Section 5), and decision procedures, such as checking and encoding equality, can be more easily automated.

In this section, we introduce the Pattern Logical Framework, called PLF. This is a uniform framework based on a dependent typed lambda calculus enriched with pattern matching in lambda abstractions. PLF can be viewed as an instance of the General Logical Framework GLF, by considering predicates corresponding to PLF patterns, similarly to what was done in Section 2 for the Rewriting Calculus. In contrast to the simple lambda calculus, the pattern-matching algorithm can either fire a substitution, or keep the computation stuck, unless further substitutions are provided. E.g., for an algebraic constant f of type $a \to a$, $M \equiv (\lambda(f\,y){:}[y{:}a].y)\,x$ is stuck, but $(\lambda(f\,x){:}[x{:}a].M)\,(f\,(f\,3)) \mapsto\!\!\!\!\to_\beta 3$. As it is well known, since the seminal work of [31], in untyped calculi, variables in patterns can be bound only if they occur *linearly* (*i.e.* at most once) and *not actively* (*i.e.* not in functional position), otherwise confluence is lost. For this reason, only *algebraic patterns* are often considered in the literature, [10, 4, 39]. The Pattern Logical Framework that we present in this section features a larger set of patterns, essentially corresponding to suitable normal forms satisfying linearity and inactivity conditions of variables. For this calculus, we show confluence, subject reduction, and strong normalization. The proof of strong normalization is technically quite difficult, and it is based on a generalized computability argument which accommodates the possibility for an argument to match the pattern after reduction.

3.1 PLF *Terms*

Since patterns occur as *binders* in abstractions, the types of the "matchable" variables in the pattern are decorated in suitable contexts, *i.e.* a pattern lambda abstraction has the form $\lambda P{:}\Delta.M$. In the following definition, we introduce the PLF pseudo-syntax for kinds, families, objects and contexts.

Definition 3.1 (PLF Pseudo-syntax)

$$
\begin{array}{rll}
\Sigma \in \mathcal{S} & \Sigma \;::=\; \emptyset \mid \Sigma, a{:}K \mid \Sigma, f{:}A & \textit{Signatures} \\
\Gamma, \Delta \in \mathcal{C} & \Gamma \;::=\; \emptyset \mid \Gamma, x{:}A & \textit{Contexts} \\
K \in \mathcal{K} & K \;::=\; \mathsf{Type} \mid \Pi P{:}\Delta.K \mid \lambda P{:}\Delta.K \mid K\,M & \textit{Kinds} \\
A, B, C \in \mathcal{F} & A \;::=\; a \mid \Pi P{:}\Delta.A \mid \lambda P{:}\Delta.A \mid A\,M & \textit{Families} \\
M, N, Q \in \mathcal{O} & M \;::=\; f \mid x \mid \lambda P{:}\Delta.M \mid M\,M & \textit{Objects}
\end{array}
$$

where $P \in \mathcal{O}_{\mathcal{P}} \subseteq \mathcal{O}$ *and* $\mathcal{O}_{\mathcal{P}}$ *is a set of* patterns *to be defined (see Definition 3.10 below).*

In a PLF pattern abstraction $\lambda P{:}\Delta.M$, P is the *pattern* to be matched, Δ is the type context containing the type of all the free variables of P, and M is the usual body of the abstraction. In a PLF pattern type-product $\Pi P{:}\Delta.A$, object dependencies are spread much more than in the standard LF. Namely, P is the *object pattern* to be matched, Δ is the type context containing the type of all the free variables of P, and A is the usual dependent type codomain, containing possibly free occurrences of some free variables of P, hence declared in Δ.

As usual, application associates to the right. Let "T" range over any term in the calculus (kind, family, object), and let the symbol "$\checkmark$" range over the set of binders $\{\lambda, \Pi\}$. 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 (corresponding to plain typed lambda calculus). As in ordinary systems dealing with dependent-types, we suppose that, in the context $\Gamma, x{:}T$, the variable x does not occur in Γ and T. $\mathsf{Dom}(\Gamma)$ and $\mathsf{CoDom}(\Gamma)$ are defined as usual. The definition of free variables needs to be rephrased as follows.

Definition 3.2 (Free Variables) *The set* Fv *of free variables in terms, signatures and contexts is given by:*

$$
\begin{array}{lcl}
\mathsf{Fv}(\emptyset), \mathsf{Fv}(\mathsf{Type}), \mathsf{Fv}(a), \mathsf{Fv}(f) & \triangleq & \emptyset \\
\mathsf{Fv}(\Sigma, a{:}K) & \triangleq & \mathsf{Fv}(\Sigma) \cup \mathsf{Fv}(K) \\
\mathsf{Fv}(\Sigma, f{:}A) & \triangleq & \mathsf{Fv}(\Sigma) \cup \mathsf{Fv}(A) \\
\mathsf{Fv}(\Delta, x{:}A) & \triangleq & \mathsf{Fv}(\Delta) \cup (\mathsf{Fv}(A) \setminus \mathsf{Dom}(\Delta)) \\
\mathsf{Fv}(x) & \triangleq & \{x\} \\
\mathsf{Fv}(\checkmark P{:}\Delta.T) & \triangleq & ((\mathsf{Fv}(P) \cup \mathsf{Fv}(T)) \setminus \mathsf{Dom}(\Delta)) \cup \mathsf{Fv}(\Delta) \\
\mathsf{Fv}(T_1\, T_2) & \triangleq & \mathsf{Fv}(T_1) \cup \mathsf{Fv}(T_2)
\end{array}
$$

Ex: $\mathsf{Fv}(\ \lambda(\lambda x{:}[x{:}\Pi w{:}a.a].x\, y){:}[y{:}a].z\ ) = \{z\}$.

We denote by $\mathsf{Bv}(T)$ the set of *bound variables* of a term T, *i.e.* the set of variables in the term which are not free. Let denote by Var the set of all variables, and by $\mathsf{Var}(T)$ the set of both free and bound variables of T. 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.

Definition 3.3 (Substitutions) *A substitution* θ *is a finite map* $[M_1/x_1, \ldots, M_m/x_m]$. *The application of a substitution* θ *to a term* T *extends the definition for the typed lambda calculus (possibly by renaming bound variables) as* $(\checkmark P{:}\Delta.M)\theta \triangleq \checkmark P\theta{:}\Delta\theta.M\theta$, *where* $\Delta\theta$ *denotes the point-wise extension of the substitution application to contexts. As usual we let* $\mathsf{Dom}(\theta) \triangleq [x_1, \ldots, x_m]$, *and*

$$\mathsf{CoDom}(\theta) \triangleq \bigcup_{i=1\ldots m} \mathsf{Fv}(M_i).$$

In what follows, we will consider only *safe terms*, *i.e.* terms where the free variables occurring in patterns are *precisely* the variables declared in the corresponding context. Formally:

Definition 3.4 (PLF Safe Terms) *A* PLF *term T is safe if* $\mathsf{EPC}(T)$ *holds, where the predicate* $\mathsf{EPC}(T)$, Exact Pattern Condition, *is defined by induction on the structure of T as follows.*

$$\begin{aligned}
\mathsf{EPC}(x) &\triangleq \mathsf{true}\\
\mathsf{EPC}(\checkmark P{:}\Delta.T) &\triangleq (\mathsf{Dom}(\Delta) = \mathsf{Fv}(P)) \wedge \mathsf{EPC}(P) \wedge \mathsf{EPC}(T) \wedge \mathsf{EPC}(\Delta)\\
\mathsf{EPC}(T_1\, T_2) &\triangleq \mathsf{EPC}(T_1) \wedge \mathsf{EPC}(T_2)
\end{aligned}$$

where $\mathsf{EPC}(\Delta)$ *holds if and only if* $\mathsf{EPC}(A)$ *holds for all* $A \in \mathsf{CoDom}(\Delta)$.

The above restriction is motivated by the fact that, if we allow free variables in patterns which are not declared in the context, we loose confluence of the untyped system (see Section 3.3 for more details). Vice versa, if we allow more variables in the context, then we loose subject reduction. Notice that substitutions applied to safe terms do not act on patterns.

We still have to specify the syntax of patterns. In order to do this, we first need to introduce the notion of matching between objects.

3.2 Matching and Operational Semantics

PLF features pattern abstractions whose application requires solving matching problems. The next two definitions introduce the notions of matching system and matching algorithm. Both are an easy modification of the ones presented in [4]. The algorithm is first-order, hence decidable.

Definition 3.5 (Matching System) (i) *A matching system*

$$\mathsf{T} \triangleq \bigwedge_{i=0\ldots n} M_i \prec^{\mathbb{V}}_{\mathbb{W}_i} N_i$$

is a conjunction of matching equations, where $\wedge$ is idempotent, associative and commutative. The set $\mathbb{V}$ records the name of the free variables that are matchable, while the sets $\mathbb{W}_i$ record the names of bound variables appearing in abstractions which cannot be matched.

(ii) *A matching system* T *is solved by the substitution θ if for all $i = 0\ldots n$, we have that $M_i\theta \equiv N_i$.*

(iii) *A matching system* T *is in normal form when it has the form*

$$\mathsf{T} \triangleq \bigwedge_{i=0\ldots m} x_i \prec^{\mathbb{V}}_{\mathbb{W}_i} N_i \;\wedge\; \bigwedge_{j=0\ldots n} f_j \prec^{\mathbb{V}}_{\mathbb{W}_j} f_j$$

(iv) *A matching system in normal form is solvable and produces the substitution* $[N_1/x_1 \cdots N_n/x_n]$ *if the following conditions are satisfied (otherwise the matching fails)*

(a) *for all* $h, k = 0 \ldots n$, *if* $x_h \equiv x_k$ *then* $N_h \equiv N_k$. *The rationale is to rule out matching-clashes, e.g.* $x \prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{W}} y \wedge x \prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{U}} z$

(b) *for all* $i = 0 \ldots n$, *if* $x_i \in \mathbb{W}_i$, *then* $N_i \equiv x_i$. *The rationale is to forbid to match a bound variable* x *against a free one* y, *e.g.* $x \prec\!\!\!\prec^{\mathbb{V}}_{x} y$

(c) *for all* $i = 0 \ldots n$, *if* $\mathsf{Fv}(N_i) \cap \mathbb{W}_i \neq \emptyset$, *then* $N_i \equiv x_i$. *The rationale is to forbid to match a free variable* x *with a bound one* y, *e.g.* $x \prec\!\!\!\prec^{\mathbb{V}}_{y} y$

Let solve *be a function that returns a substitution if a matching system in normal form is solvable, and fails otherwise, i.e.*

$$\mathsf{solve}(\mathsf{T}) = \begin{cases} \theta & \text{if T is solvable with } \theta \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Definition 3.6 (Matching Algorithm $\mathcal{A}lg$) (i) *The reduction* $\rightsquigarrow$ *is the compatible relation induced by the following two rules:*

$$\frac{}{M_1\, N_1 \;\prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{U}}\; M_2\, N_2 \;\rightsquigarrow\; M_1 \;\prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{U}}\; M_2 \wedge N_1 \;\prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{U}}\; N_2}\ (Appl)$$

$$\frac{\mathbb{W} \triangleq \mathbb{U} \cup \mathsf{Dom}(\Delta)}{\checkmark P{:}\Delta.T_1 \;\prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{U}}\; \checkmark P{:}\Delta.T_2 \;\rightsquigarrow\; T_1 \;\prec\!\!\!\prec^{\mathbb{V}}_{\mathbb{W}}\; T_2}\ (Lbd/Prod)$$

In rule (Lbd/Prod), *the condition* $\mathbb{W} \triangleq \mathbb{U} \cup \mathsf{Dom}(\Delta)$ *increases the set of bound variables to be matched; moreover, since all free variables in* P *are declared in the context* Δ, *two abstraction/product terms match if and only if they have the same pattern (up-to* α*-conversion).*

(ii) *The reduction* $\rightsquigarrow^*$ *is defined as the reflexive and transitive closure of* $\rightsquigarrow$. *Let* norm *be the function that reduces a matching system in normal form, or fails, i.e.*

$$\mathsf{norm}\ (\mathsf{T}) \triangleq \begin{cases} \mathsf{T}' & \text{if } \mathsf{T} \rightsquigarrow^* \mathsf{T}' \text{ and } \mathsf{T}' \text{ is in normal form} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

(iii) *Let* $\mathcal{A}lg(M;N)$ *be defined as follows.*

$$\mathcal{A}lg(M;N) \triangleq \begin{cases} \mathsf{fail} & \text{if } \mathsf{solve}(\mathsf{norm}(M \;\prec\!\!\!\prec^{\mathsf{Fv}(M)}_{\emptyset}\; N)) = \mathsf{fail} \\ \mathsf{solve}(\mathsf{norm}(M \;\prec\!\!\!\prec^{\mathsf{Fv}(M)}_{\emptyset}\; N)) & \text{otherwise} \end{cases}$$

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

The matching algorithm $\mathcal{A}lg$ is sound, in the sense that, if the initial matching system is solvable, then the substitution computed by $\mathcal{A}lg$ solves this system.

Lemma 3.7 (Soundness of $\mathcal{A}lg$) *If* $\mathcal{A}lg(M;N) = \theta$, *then* $M\theta \equiv N$.

The next definition introduces the standard notions of one-step, many-steps β-reduction, and the corresponding congruence relation.

Definition 3.8 (One/Many-Steps Reduction, Congruence) *Let* $\theta = \mathcal{A}lg(P;N)$.

(i) *The top-level rules are*

$$(\beta\text{−Obj}) \qquad (\lambda P{:}\Delta.M)\, N \mapsto_\beta M\theta$$

$$(\beta\text{−Fam}) \qquad (\lambda P{:}\Delta.A)\, N \mapsto_\beta A\theta$$

$$(\beta\text{−Kinds}) \qquad (\lambda P{:}\Delta.K)\, N \mapsto_\beta K\theta$$

(ii) *Let* $\mathsf{C}[-]$ *denote a pseudo-context with a "single hole" inside, defined on terms and contexts as follows*

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

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 rule*

$$\frac{T_1 \mapsto_\beta T_2}{\mathsf{C}[T_1] \mapsto_\beta \mathsf{C}[T_2]}\ (Ctx)$$

(iii) *The many-step evaluation* $\mapsto\!\!\!\!\rightarrow_\beta$ *and the congruence relation* $=_\beta$ *are defined respectively as the reflexive-transitive and reflexive-symmetric-transitive closure of* $\mapsto_\beta$. *By* $\mapsto_\beta^0$ *we denote the reflexive closure of* $\mapsto_\beta$.

3.3 PLF *Patterns*

In this subsection, we will characterize the set of patterns in $\mathcal{O}_\mathcal{P}$, which we left unspecified in Definition 3.10. Such patterns will be objects in suitable normal form, satisfying the following conditions:

- each free variable appears at most once (linearity condition);
- variables are not in functional position (non-activity condition).

The notion of normal form which we consider requires special care. Namely: terms are taken to be in normal form whenever all redexes are *substitution-stuck*, *i.e.* they are stuck, no matter what substitution is applied to the argument, formally:

Definition 3.9 (PLF Normal Forms) PLF *contexts and terms in normal form are*

mutually defined as follows.

$$\mathsf{Nf}_{\mathcal{C}} \ni \quad \Gamma ::= \emptyset \mid \Gamma, x{:}A$$

$$\mathsf{Nf}_{\mathcal{K}} \ni \quad K ::= \mathsf{Type}\, K_1 \ldots K_n \mid \Pi P{:}\Delta.K \mid (\lambda P{:}\Delta.K)\, N\, K_1 \ldots K_n$$

$$\mathsf{Nf}_{\mathcal{F}} \ni \quad A ::= a\, A_1 \ldots A_n \mid \Pi P{:}\Delta.A \mid (\lambda P{:}\Delta.A)\, N\, A_1 \ldots A_n$$

$$\mathsf{Nf}_{\mathcal{O}} \ni M, N ::= f\, M_1 \ldots M_n \mid x\, M_1 \ldots M_n \mid \lambda P{:}\Delta.M \mid (\lambda P{:}\Delta.M)\, N\, M_1 \ldots M_n$$

where, the redexes $(\lambda P{:}\Delta.K)\, N$, *and* $(\lambda P{:}\Delta.A)\, N$, *and* $(\lambda P{:}\Delta.M)\, N$ *are "substitution-stuck", i.e., for any substitution* θ, $\mathcal{A}lg(P; N\theta) = \mathsf{fail}$.

Finally, we are in the position of characterizing the set of patterns in $\mathcal{O}_{\mathcal{P}}$:

Definition 3.10 (PLF Patterns) *Let* $\mathcal{O}_{\mathcal{P}}$ *be the set of objects defined by*

$$\mathcal{O}_{\mathcal{P}} \triangleq \{P \in \mathsf{Nf}_{\mathcal{O}} \mid \textsc{lpc}(P; \mathsf{Fv}(P)) = \mathsf{true} \wedge \textsc{apc}(P; \mathsf{Fv}(P)) = \mathsf{false}\}$$

where, for any term T *and finite set of variables* $\mathbb{V}$,

- *the predicate* $\textsc{lpc}(T; \mathbb{V})$, Linear Pattern Condition, *is defined by induction on* T *as follows.*

$$\begin{aligned}
\textsc{lpc}(x/f/a; \mathbb{V}) &\triangleq \mathsf{true}\\
\textsc{lpc}(\checkmark P{:}\Delta.T; \mathbb{V}) &\triangleq \textsc{lpc}(P; \mathsf{Dom}(\Delta)) \wedge \textsc{lpc}(\Delta; \mathbb{V} \cup \mathsf{Dom}(\Delta)) \wedge \textsc{lpc}(T; \mathbb{V} \cup \mathsf{Dom}(\Delta))\\
\textsc{lpc}(T_1\, T_2; \mathbb{V}) &\triangleq \textsc{lpc}(T_1; \mathbb{V}) \wedge \textsc{lpc}(T_2; \mathbb{V}) \wedge (\mathsf{Fv}(T_1) \cap \mathsf{Fv}(T_2) \cap \mathbb{V} = \emptyset)
\end{aligned}$$

- *the predicate* $\textsc{apc}(T; \mathbb{V})$, Active Pattern Condition, *is defined by induction on* T *as follows.*

$$\begin{aligned}
\textsc{apc}(x/f/a; \mathbb{V}) &\triangleq \mathsf{false}\\
\textsc{apc}(\checkmark P{:}\Delta.T; \mathbb{V}) &\triangleq (P \equiv x\, P_1 \wedge x \in \mathsf{Dom}(\Delta)) \vee \textsc{apc}(P; \mathsf{Dom}(\Delta)) \vee\\
&\qquad \textsc{apc}(T; \mathbb{V} \cup \mathsf{Dom}(\Delta)) \vee \textsc{apc}(\Delta; \mathbb{V} \cup \mathsf{Dom}(\Delta))\\
\textsc{apc}(T_1\, T_2; \mathbb{V}) &\triangleq \textsc{apc}(T_1; \mathbb{V}) \vee \textsc{apc}(T_2; \mathbb{V})
\end{aligned}$$

At first sight, the above definitions of normal forms and patterns may seem a little awkward, because of the requirement that only those redexes are considered, which are stuck no matter what substitution is applied to the argument. Somewhat surprisingly, such a restriction is necessary to achieve confluence. Actually, any of the restrictions on patterns imposed in Definition 3.10 above can be hardly relaxed, apart from considering only well-typed terms. In the following, we discuss in detail each condition, and possible extensions.

(i) *Variables in functional position.* It is well known, since [31], that allowing variables in functional position breaks confluence. Here is a simple counterexample: $M \triangleq (\lambda(x\,y){:}[x{:}a{\to}a, y{:}a].x)\,(\mathsf{I}\,z)$, where $\mathsf{I} \triangleq \lambda x{:}a.x$. Namely, $M \mapsto_\beta (\lambda(x\,y){:}[x{:}a{\to}a, y{:}a].x)\,z$, by reducing the argument, while $M \mapsto_\beta \mathsf{I}$, by reducing the outermost redex.

(ii) *Linearity condition.* Since [31], it is also well-known that if we abandon the linearity condition in patterns, we loose confluence of raw terms (*i.e.* all PLF terms, including also terms not typable in the type system of Section 3.4 below). Namely, let

- $Y \triangleq (\lambda y{:}?.\lambda x{:}?.(x\,(y\,y\,x)))\,(\lambda y{:}?.\lambda x{:}?.(x\,(y\,y\,x)))$ be the (hopefully untypable) fix-point combinator
- $N \triangleq \lambda(f\,z\,z){:}[z{:}a].g$ be a term with a non-linear pattern
- $M \triangleq Y(\lambda y{:}?.\lambda x{:}?.N\,(f\,x\,(y\,x)))$
- $Q \triangleq Y\,M$

Then, we have $Q \mapsto\!\!\!\!\to_\beta C\,g$, and $Q \mapsto\!\!\!\!\to_\beta g$. Thus the system is not confluent. However, one can check that the fix-point operator Y is *not* typable in the PLF type system of Section 3.4 below. Hence the above counterexample does not apply to the case of well-typed terms. Actually, we do not know whether the linearity condition may be relaxed, without loosing confluence of well-typed terms. In this paper, we stick with this condition, and we prove confluence for all raw terms.

(iii) *Substitution-stuck redexes.* The reason for allowing in patterns only substitution-stuck redexes, and not simply stuck redexes, is that, in this way, patterns can match only arguments where the corresponding redexes will never fire. Otherwise, if we include patterns of the shape $(\lambda P_1{:}\Delta.P_2)\,P_3\,\boldsymbol{P'}$, where only $\mathcal{A}lg(P_1; P_3) = \mathsf{fail}$, *i.e.* only the present reduction is stuck, we loose confluence. The following term gives a counterexample $M \triangleq (\lambda((\lambda\mathsf{I}{:}\emptyset.\mathsf{I})\,x){:}[x{:}a{\to}a].x)\,((\lambda\mathsf{I}{:}\emptyset.\mathsf{I})\,\mathsf{I})$. Namely, by reducing the outermost redex, we get $M \mapsto_\beta \mathsf{I}$; while, by reducing inside the argument, $M \mapsto_\beta (\lambda((\lambda\mathsf{I}{:}\emptyset.\mathsf{I})\,x){:}[x{:}a{\to}a].x)\,\mathsf{I}$.

(iv) *Exact Pattern Condition.* In this paper, we consider only terms where the variables occurring in patterns are precisely the variables declared in the corresponding contexts. Namely, by relaxing this condition to $\mathsf{Fv}(P) \subseteq \mathsf{Dom}(\Delta)$, we loose subject reduction. E.g., from $x{:}A \vdash (\lambda z{:}[z{:}A, y{:}B].y)\,x : (\lambda z{:}\Delta.B)\,x$, by reducing both the term and the type, we have $x{:}A \vdash y{:}B$, which is not derivable. On the other hand, one could think of having $\mathsf{Dom}(\Delta) \subseteq \mathsf{Fv}(P)$, *i.e.* patterns can contain free variables, which can be bound outside, and hence they can be substituted during reductions, as the variable y in the following term $(\lambda y{:}a.\lambda(f\,x\,y){:}[x{:}a].y)\,z \mapsto_\beta \lambda(f\,x\,z){:}[x{:}a].z$ But this causes problems when combined with untypable fix-points, since, as noticed in [39], the non-linear term N in item (ii) above can be mimicked in this setting, even under the linearity pattern condition. Namely, let $M \triangleq \lambda x{:}a.\lambda x{:}\emptyset.g$. Then M behaves as N of item (ii), since $M\,N_1\,N_2 \mapsto_\beta (\lambda N_1{:}\emptyset.g)\,N_2 \mapsto_\beta g$ if and only if $N_1 \equiv N_2$. Thus M, combined with the untypable fix-point operator Y, breaks confluence of raw

terms.

(v) *Pattern reductions.* The counterexample in item (iii) above also shows that extending the class of patterns beyond normal forms, by allowing reductions in patterns is potentially dangerous. In this perspective, in order to preserve confluence when reductions in patterns are permitted, a possible solution is that of allowing reductions to fire only when the pattern is a normal form in the sense of Definition 3.10. This corresponds to partially fixing a reduction strategy. However, *K-reductions* in patterns deserve special discussion.

(vi) *K-reductions in patterns.* A *K-redex* is a redex $(\lambda P{:}\Delta.M)\,N$, where $\lambda P{:}\Delta.M$ is a *K-abstraction*, *i.e.* $\mathsf{Fv}(M) \subset \mathsf{Fv}(P)$. When a K-redex is reduced, (parts of) the argument is erased. As a consequence, the Exact Pattern Condition is violated, and bound variables may become free. Here is an example:
$M \triangleq (\lambda \underbrace{((\lambda x{:}a.y)\,z)}_{P}{:}[y{:}a{\to}a, z{:}a].y\,z)\,\underbrace{((\lambda x{:}a.f)\,g)}_{N}$ Then, by reducing the pattern P and the argument N, and then reducing the outermost redex, we get $M \mapsto\!\!\!\!\to_\beta (\lambda y{:}[y{:}a{\to}a, z{:}a].y\,z)\,f \mapsto_\beta f\,z$, *i.e.* z comes out of its scope!
To avoid this problem, we could simply block K-reductions in patterns, but then we also need to block pattern matching when the pattern contains a K-redex. Otherwise, we loose confluence, the term M above being a counterexample. Namely, by reducing the outermost redex, $M \mapsto_\beta f\,g$, while, by reducing the argument N, we get $M \mapsto_\beta (\lambda((\lambda x{:}a.y)\,z){:}[y{:}a{\to}a, z{:}a].y\,z)\,f$, which is not reducible anymore.

The above discussion shows that reaching confluence regardless typability is a rather brittle property, and can be lost even for small extensions of the definition of patterns. On the basis of all this, in Definition 3.10 above, we have carefully devised a notion of pattern, and corresponding reduction, which we will see satisfies the confluence property, but nevertheless is considerably general. In our case, confluence holds already for raw terms. This turns out to be particularly handy in proving strong normalization.

In particular, our definition of patterns guarantees the validity of the Matching Preservation Lemma and the Substitution Lemma below, which are crucial for proving confluence and some fundamental properties of the PLF type system, such as subject reduction and strong normalization.

The Matching Preservation Lemma (which can be proved by induction on patterns) expresses the fact that matchings are preserved both under $\mapsto_\beta$-reductions, and substitutions of the argument, *i.e.*:

Lemma 3.11 (Reduction/Substitution Preserve Matching) (i) *If $\mathcal{A}lg(P;N) = \theta$ and $N \mapsto_\beta N'$, then there exist θ' such that $\mathcal{A}lg(P;N') = \theta'$ and $\theta \mapsto\!\!\!\!\to_\beta \theta'$;*

(ii) *If $\overline{\theta} = \mathcal{A}lg(P;N)$, then, for all θ such that $\mathsf{Var}(\overline{\theta}) \cap \mathsf{CoDom}(\theta) = \emptyset$, there exists $\overline{\theta}' = \mathcal{A}lg(P;N\theta)$; moreover, for all T, we have $T\overline{\theta}\theta \equiv T\theta\overline{\theta}'$.*

Using Lemma 3.11(ii), we can prove:

Lemma 3.12 (Substitution) *If $T \mapsto\!\!\!\!\rightarrow_\beta T'$ and $\theta \mapsto\!\!\!\!\rightarrow_\beta \theta'$, then $T\theta \mapsto\!\!\!\!\rightarrow_\beta T'\theta'$.*

The proof of confluence is a suitable application of the usual argument based on *parallel reduction* of [37]. As pointed out above, confluence holds for raw terms, provided they satisfy the suitable restrictions on patterns introduced so far.

Theorem 3.13 (Confluence) *The relation $\mapsto_\beta$ is confluent.* □

3.4 PLF *Type System*

Signatures rules

$$\frac{}{\emptyset\ \mathsf{sig}}\ (S{\cdot}Empty)$$

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K\ \mathsf{sig}}\ (S{\cdot}Kind)$$

$$\frac{\Sigma\ \mathsf{sig} \quad \vdash_\Sigma A : \mathsf{Type} \quad f \notin \mathsf{Dom}(\Sigma)}{\Sigma, f{:}A\ \mathsf{sig}}\ (S{\cdot}Type)$$

Contexts rules

$$\frac{\Sigma\ \mathsf{sig}}{\vdash_\Sigma \emptyset}\ (C{\cdot}Empty)$$

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

Kind rules

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

$$\frac{\Gamma, \Delta \vdash_\Sigma P : A \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi P{:}\Delta.K}\ (K{\cdot}Pi)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma P : A \quad \Gamma, \Delta \vdash_\Sigma K}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.K}\ (K{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma \Pi P{:}\Delta.K \quad \Gamma, \Delta \vdash_\Sigma P : A \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma (\lambda P{:}\Delta.K)\, N}\ (K{\cdot}Appl)$$

Families rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Gamma}{\Gamma \vdash_\Sigma a : K}\ (F{\cdot}Var)$$

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

$$\frac{\Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma, \Delta \vdash_\Sigma A : K}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A : \Pi P{:}\Delta.K}\ (F{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma A : \Pi P{:}\Delta.K \quad \Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma \vdash_\Sigma N : B}{\Gamma \vdash_\Sigma A\, N : (\lambda P{:}\Delta.K)\, N}\ (F{\cdot}Appl)$$

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

Object rules

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

$$\frac{\vdash_\Sigma \Gamma \quad f{:}A \in \Sigma}{\Gamma \vdash_\Sigma f : A}\ (O{\cdot}Const)$$

$$\frac{\Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma, \Delta \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.M : \Pi P{:}\Delta.A}\ (O{\cdot}Abs)$$

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

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

Figure 3. The PLF Type System

PLF involves type judgments of the following shape:

$$\Sigma\ \mathsf{sig} \qquad (\Sigma \text{ is a valid signature})$$

$$\vdash_\Sigma \Gamma \qquad (\Gamma \text{ is a valid context in } \Sigma)$$

$$\Gamma \vdash_\Sigma K \qquad (K \text{ is a kind in } \Gamma \text{ and } \Sigma)$$

$$\Gamma \vdash_\Sigma A : \mathsf{Type} \qquad (A \text{ is has kind } K \text{ in } \Gamma \text{ and } \Sigma)$$

$$\Gamma \vdash_\Sigma M : A \qquad (M \text{ is has type } A \text{ in } \Gamma \text{ and } \Sigma)$$

The typing rules of PLF are presented in Figure 3. As remarked in the introduction, rules (F·Appl), (O·Appl) do not utilize metasubstitution as in standard LF, but rather introduce an explicit type redex. Rules (F·Conv), and (O·Conv) allow to recover the usual rules, if the reduction fires.

Strictly speaking, one should mention also the auxiliary equality judgments, but in view of the fact that confluence holds also over non well-typed terms, we do not need contexts and signatures in the equality judgments, and therefore they can be safely "swept under the rug".

Let $\Gamma \vdash_\Sigma \alpha$ be any judgment in the system. Lemmas 3.14, 3.15, 3.16, 3.17 below are the instantiations of Conjecture 2.8 to PLF.

Lemma 3.14 (Subderivation Property) • *Any derivation of* $\Gamma \vdash_\Sigma \alpha$ *has subderivations of* Σ sig *and* $\vdash_\Sigma \Gamma$*;*

- *Any derivation of* $\Sigma, a{:}K$ sig *has subderivations of* Σ sig *and* $\vdash_\Sigma K$*;*
- *Any derivation of* $\Sigma, f{:}A$ sig *has subderivations of* Σ sig *and* $\vdash_\Sigma A : \mathsf{Type}$*;*
- *Any derivation of* $\vdash_\Sigma \Gamma, x{:}A$ *has subderivations of* Σ sig *and* $\Gamma \vdash_\Sigma A : \mathsf{Type}$*;*
- *Given a derivation of* $\Gamma \vdash_\Sigma \alpha$ *and any subterm occurring in the subject of the judgment, there exists a derivation of a smaller length of a judgment having that subterm as a subject;*
- *If* $\Gamma \vdash_\Sigma A : K$*, then* $\Gamma \vdash_\Sigma K$*;*
- *If* $\Gamma \vdash_\Sigma M : A$*, then* $\Gamma \vdash_\Sigma A : \mathsf{Type}$.

Lemma 3.15 (Permutation) *If* $\Gamma_1, x{:}A, \Delta, y{:}B, \Gamma_2 \vdash_\Sigma \alpha$*, then* $\Gamma_1, y{:}B, \Delta, x{:}A, \Gamma_2 \vdash_\Sigma \alpha$*, provided that* $x \notin \mathsf{Fv}(\Delta) \cup \mathsf{Fv}(B)$.

Lemma 3.16 (Weakening) *If* $\Gamma \vdash_\Sigma \alpha$ *and* $\vdash \Gamma, \Delta$*, then* $\Gamma, \Delta \vdash_\Sigma \alpha$.

Lemma 3.17 (Unicity of Types and Kinds) *If* $\Gamma \vdash_\Sigma T : T_1$ *and* $\Gamma \vdash_\Sigma T : T_2$*, then* $\Gamma \vdash_\Sigma T_1 =_\beta T_2$.

Lemma 3.18 (Transitivity) *If* $\Gamma, x{:}A, \Delta \vdash_\Sigma \alpha$ *and* $\Gamma \vdash_\Sigma M : A$*, then* $\Gamma, \Delta[M/x] \vdash_\Sigma \alpha[M/x]$.

Lemma 3.19 (Abstraction Typing) • *If* $\Gamma \vdash_\Sigma T$ *(or* $\Gamma \vdash_\Sigma T : T'$*) and* Γ' *is such that* $\mathsf{Dom}(\Gamma) = \mathsf{Dom}(\Gamma')$*, and for all* $x \in \mathsf{Dom}(\Gamma)$*,* $\Gamma \vdash_\Sigma \Gamma(x) =_\beta \Gamma'(x)$ *and* $\mathsf{Fv}(\Gamma(x)) \subseteq \mathsf{Fv}(\Gamma'(x))$*, then* $\Gamma' \vdash_\Sigma T'$ *(or* $\Gamma' \vdash_\Sigma T : T'$*);*

- *If* $\Gamma \vdash_\Sigma \lambda P{:}\Delta.T : \Pi P'{:}\Delta'.T'$*, then* $\mathsf{Dom}(\Delta) = \mathsf{Dom}(\Delta')$*, and for all* $x \in \mathsf{Dom}(\Delta)$*, we have* $\Gamma, \Delta \vdash_\Sigma \Delta(x) =_\beta \Delta'(x)$*, and* $\Gamma, \Delta \vdash_\Sigma P =_\beta P'$*;*
- *If* $\Gamma \vdash_\Sigma \lambda P{:}\Delta.T : \Pi P{:}\Delta.T'$*, then* $\Gamma, \Delta \vdash_\Sigma P : \sigma$ *and* $\Gamma, \Delta \vdash_\Sigma T : T'$.

We are now ready to prove that typing is preserved by reduction.

Theorem 3.20 (Subject Reduction) (i) *If* $\Gamma \vdash_\Sigma K$ *and* $K \mapsto_\beta K'$, *then* $\Gamma \vdash_\Sigma K'$.

(ii) *If* $\Gamma \vdash_\Sigma A : K$ *and* $A \mapsto_\beta B$, *then* $\Gamma \vdash_\Sigma B : K$;

(iii) *If* $\Gamma \vdash_\Sigma M : A$ *and* $M \mapsto_\beta N$, *then* $\Gamma \vdash_\Sigma N : A$.

3.5 Strong Normalization

Let $\mathsf{SN} = \mathsf{SN}^{\mathcal{O}} \cup \mathsf{SN}^{\mathcal{F}} \cup \mathsf{SN}^{\mathcal{K}}$ be the set of strongly normalizing terms. This section is devoted to the proof of the following theorem:

Theorem 3.21 (Strong Normalization) (i) *If* $\Gamma \vdash_\Sigma K$, *then* $K \in \mathsf{SN}^{\mathcal{K}}$;

(ii) *If* $\Gamma \vdash_\Sigma A : K$, *then* $A \in \mathsf{SN}^{\mathcal{F}}$;

(iii) *If* $\Gamma \vdash_\Sigma M : A$, *then* $M \in \mathsf{SN}^{\mathcal{O}}$.

The proof of the above theorem is based on a non-trivial extension of the standard *Computability Argument* to accommodate the presence of patterns in the syntax. For technical reasons, in this section we find convenient to work in the equivalent PLF system with the more informative lambda pattern abstraction $\checkmark P{:}\Delta{:}B.T$, where B is meant to be the type inferred for P. We will omit B when it is irrelevant in proofs.

Definition 3.22 (Comp Sets) • *Let* $\mathsf{Comp}^{\mathcal{O}}$ *be the set of* object computability candidates *defined as follows.*
$\mathcal{N} \in \mathsf{Comp}^{\mathcal{O}}$ *if and only if* $\mathcal{N}$ *satisfies:*
(c1) $\mathcal{N} \subseteq \mathsf{SN}^{\mathcal{O}}$;
(c2) $\forall \boldsymbol{N} \in \mathsf{SN}^{\mathcal{O}}.\ x\,\boldsymbol{N}$, *and* $f\,\boldsymbol{N} \in \mathcal{N}$;
(c3) $\mathcal{N}$ *is closed under the rule*

$$\frac{Q \mapsto\!\!\!\!\rightarrow_\beta Q' \quad \mathcal{A}lg(P;Q') = \theta \quad (M\theta)\boldsymbol{N} \in \mathcal{N} \quad \mathsf{CoDom}(\Delta), Q \in \mathsf{SN}}{(\lambda P{:}\Delta.M)\,Q\,\boldsymbol{N} \in \mathcal{N}}$$

(c4) $\mathcal{N}$ *is closed under the rule*

$$\frac{\forall Q'.\ [Q \mapsto\!\!\!\!\rightarrow_\beta Q' \Rightarrow \mathcal{A}lg(P;Q') = \mathsf{fail}] \quad \mathsf{CoDom}(\Delta), M, Q, \boldsymbol{N} \in \mathsf{SN}}{(\lambda P{:}\Delta.M)\,Q\,\boldsymbol{N} \in \mathcal{N}}$$

• *Let* $\mathsf{Comp}^{\mathcal{F}}$ *be the set of* family computability candidates *defined as follows.*
$\mathcal{N} \in \mathsf{Comp}^{\mathcal{F}}$ *if and only if* $\mathcal{N}$ *satisfies:*
(c1) $\mathcal{N} \subseteq \mathsf{SN}^{\mathcal{F}}$;
(c2) $\forall \boldsymbol{N} \in \mathsf{SN}^{\mathcal{F}}.\ a\,\boldsymbol{N} \in \mathcal{N}$;
(c3) $\mathcal{N}$ *is closed under the rule*

$$\frac{Q \mapsto\!\!\!\!\rightarrow_\beta Q' \quad \mathcal{A}lg(P;Q') = \theta \quad (A\theta)\boldsymbol{N} \in \mathcal{N} \quad \mathsf{CoDom}(\Delta), Q \in \mathsf{SN}}{(\lambda P{:}\Delta.A)\,Q\,\boldsymbol{N} \in \mathcal{N}}$$

(c4) $\mathcal{N}$ *is closed under the rule*

$$\frac{\forall Q'.\ [Q \mapsto\!\!\!\!\rightarrow_\beta Q' \Rightarrow \mathcal{A}lg(P;Q') = \mathsf{fail}] \quad \mathsf{CoDom}(\Delta), A, Q, \boldsymbol{N} \in \mathsf{SN}}{(\lambda P{:}\Delta.A)\, Q\, \boldsymbol{N} \in \mathcal{N}}$$

The rule in (**c3**) above captures the case when there exists, *eventually* a possible matching between the pattern and the argument, while the rule in (**c4**) captures the case when *never* there will be a matching. In what follows, we denote by $P \sqsubseteq Q$ the fact that there exist Q', θ such that $Q \mapsto\!\!\!\!\rightarrow_\beta Q'$ and $\theta = \mathcal{A}lg(P;Q')$, and by $P \not\sqsubseteq Q$ the fact that, for all Q' such that $Q \mapsto\!\!\!\!\rightarrow_\beta Q'$, we have $\mathcal{A}lg(P;Q') = \mathsf{fail}$.
The following lemma holds.

Lemma 3.23 $\mathsf{SN}^{\mathcal{O}} \in \mathsf{Comp}^{\mathcal{O}}$ *and* $\mathsf{SN}^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{F}}$.

The next definition, together with Lemma 3.25 below, give an interpretation of families in $\mathsf{Comp}^{\mathcal{O}}$, and of kinds in $\mathsf{Comp}^{\mathcal{F}}$. Such interpretation is defined by induction on families and kinds. The complexity measure m for families and kinds is given by the number of family/kind metaoperators like, *e.g.* ✓ and the hidden application metaoperator, *i.e.*:

$$\mathsf{m}(a) = 0 \quad \mathsf{m}(\mathsf{Type}) = 0 \quad \mathsf{m}(T\,M) = \mathsf{m}(T) + 1 \quad \mathsf{m}(\checkmark P{:}\Delta.T) = \mathsf{m}(T) + 1$$

Notice that, in particular, A and $A\theta$ have the same complexity.

Definition 3.24 (Family and Kind Interpretation) • *Let* $[\![-]\!]^{\mathcal{F}}$ *be the family interpretation function defined by induction on families as follows.*

$$[\![\, a\, \boldsymbol{N} \,]\!]^{\mathcal{F}} = \mathsf{SN}^{\mathcal{O}} \qquad [\![\, \checkmark P{:}\Delta{:}B.A \,]\!]^{\mathcal{F}} =$$
$$\left\{ M \,\middle|\, Q \in [\![B]\!]^{\mathcal{F}} \Longrightarrow M\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{[\![A\theta]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q' \wedge \theta = \mathcal{A}lg(P;Q')\} & \text{otherwise} \end{cases} \right\}$$
$$[\![\, (\lambda P{:}\Delta.A)\, M\, \boldsymbol{N} \,]\!]^{\mathcal{F}} = \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq M \\ \bigcup\{[\![(A\theta)\boldsymbol{N}]\!]^{\mathcal{F}} \mid M \mapsto\!\!\!\!\rightarrow_\beta M' \wedge \theta = \mathcal{A}lg(P;M')\} & \text{otherwise} \end{cases}$$

• *Let* $[\![-]\!]^{\mathcal{K}}$ *be the family interpretation function defined by induction on kinds as follows.*

$$[\![\, \mathsf{Type}\, \boldsymbol{N} \,]\!]^{\mathcal{K}} = \mathsf{SN}^{\mathcal{F}} \qquad [\![\, \checkmark P{:}\Delta{:}B.K \,]\!]^{\mathcal{K}} =$$
$$\left\{ A \,\middle|\, Q \in [\![B]\!]^{\mathcal{K}} \Longrightarrow A\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{F}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{[\![K\theta]\!]^{\mathcal{K}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q' \wedge \theta = \mathcal{A}lg(P;Q')\} & \text{otherwise} \end{cases} \right\}$$
$$[\![\, (\lambda P{:}\Delta.K)\, M\, \boldsymbol{N} \,]\!]^{\mathcal{K}} = \begin{cases} \mathsf{SN}^{\mathcal{F}} & \text{if } P \not\sqsubseteq M \\ \bigcup\{[\![(K\theta)\, \boldsymbol{N}]\!]^{\mathcal{K}} \mid M \mapsto\!\!\!\!\rightarrow_\beta M' \wedge \theta = \mathcal{A}lg(P;M')\} & \text{otherwise} \end{cases}$$

Then the following lemmas hold:

Lemma 3.25 (i) *For every family* A*, we have* $[\![A]\!]^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{O}}$*;*
(ii) *For every kind* K*, we have* $[\![K]\!]^{\mathcal{K}} \in \mathsf{Comp}^{\mathcal{F}}$.

Lemma 3.26 (Soundness of $[\![\ ]\!]^{\mathcal{F}}/[\![\ ]\!]^{\mathcal{K}}$ **)** (i) *If* $A \mapsto\!\!\!\!\rightarrow_\beta B$*, then* $[\![A]\!]^{\mathcal{F}} = [\![B]\!]^{\mathcal{F}}$*;*
(ii) *If* $K \mapsto\!\!\!\!\rightarrow_\beta K'$*, then* $[\![K]\!]^{\mathcal{K}} = [\![K']\!]^{\mathcal{K}}$.

Lemma 3.27 (Key Lemma) *Let* Γ *be a context, and let* $N_i \in [\![\Gamma(x_i)]\!]^{\mathcal{F}}$*, for all* $x_i \in \mathsf{Dom}(\Gamma)$*. Then:*

(i) *If* $\Gamma \vdash_\Sigma K$, *then* $K[\boldsymbol{N}/\boldsymbol{x}] \in \mathsf{SN}^{\mathcal{K}}$;

(ii) *If* $\Gamma \vdash_\Sigma A : K$, *then* $A[\boldsymbol{N}/\boldsymbol{x}] \in [\![K[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{K}}$;

(iii) *If* $\Gamma \vdash_\Sigma M : A$, *then* $M[\boldsymbol{N}/\boldsymbol{x}] \in [\![A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$.

By Lemma 3.27, using the fact that variables belong to any set in $\mathsf{Comp}^{\mathcal{O}}$, we can prove the Strong Normalization Theorem 3.21.

Finally, we are in the position of proving that PLF can be used as a framework for proof checking.

Theorem 3.28 (Judgements decidability) *It is decidable whether the* PLF *judgment* $\Gamma \vdash_\Sigma \alpha$ *is derivable.*

4 The Closed Logical Framework

In this section, we investigate the Closed Logical Framework, CLF, introduced in Section 2.4 as an instance of GLF. We recall that CLF is obtained from GLF by considering the set $\mathbb{S}_{\mathrm{CLF}} \triangleq \{ (\mathrm{True}_x, [x{:}\sigma], \overline{\mathrm{True}_x}, \widehat{\mathrm{True}_x}), (\mathrm{Closed}_x, [x{:}\sigma], \overline{\mathrm{Closed}_x}, \widehat{\mathrm{Closed}_x}) \}$. This instantiation of GLF amounts to a logical framework which features the standard β-rule as well as a restricted β-rule that fires only when the argument is closed. In Section 5, we will provide a very interesting application of CLF as a Logical Framework.

The Closed Logical Framework is an example of an interesting class of Logical Frameworks, which arise when we instantiate GLF to systems which feature standard β-reduction together with a restricted β-reduction *i.e.*

$$(\beta_{\mathsf{v}}) \qquad (\lambda x.M)\, N \to_{\beta_{\mathsf{v}}} M[N/X] \qquad \text{provided } N \in \mathcal{V}$$

where $\mathcal{V}$ is a set of *values*. Gordon Plotkin was the first to introduce this kind of restriction in the call-by-value lambda calculus, [35], in order to discuss the observational equivalence of the SECD machine. Other restricted lambda calculi were introduced in the literature, to analyze the behavior of special classes of terms, *i.e.* strongly normalizing terms. However the simultaneous combination of both the standard β and β_{v} was rarely discussed, let alone in a typed context. Once again we point out that the special nature of the type system, which records potential reductions which have not yet fired, is the crucial ingredient, which makes this enterprise worthwhile.

It is interesting to point out that, in what follows, everything goes through, provided the set $\mathcal{V}$ of values is closed under standard β-reduction and non-overlapping substitutions which derive from the reductions involved, *i.e.* provided the appropriate form of Lemma 4.3 below holds.

In discussing CLF, for the sake of brevity, we write Closed_x by $x_\emptyset$ and True_x by x. We also let $\overline{x} \in \{x, x_\emptyset\}$.

4.1 CLF *Terms*

In the next definition, we introduce the pseudo-syntax for kinds, families, objects and contexts.

Definition 4.1 (CLF Pseudo-syntax)

$$
\begin{array}{rll}
\Sigma \in \mathcal{S} & \Sigma ::= \emptyset \mid \Sigma, a{:}K \mid \Sigma, f{:}A & \textit{Signatures} \\
\Gamma, \Delta \in \mathcal{C} & \Gamma ::= \emptyset \mid \Gamma, \overline{x}{:}A & \textit{Contexts} \\
K \in \mathcal{K} & K ::= \mathsf{Type} \mid \Pi\overline{x}{:}A.K \mid \lambda\overline{x}{:}A.K \mid K\, M & \textit{Kinds} \\
A, B, C \in \mathcal{F} & A ::= a \mid \Pi\overline{x}{:}A.B \mid \lambda\overline{x}{:}A.B \mid A\, M & \textit{Families} \\
M, N, Q \in \mathcal{O} & M ::= f \mid x \mid \lambda\overline{x}{:}A.N \mid M\, N & \textit{Objects}
\end{array}
$$

4.2 *Operational Semantics*

Definition 4.2 (One/Many-Steps, Congruence) *Let $\mathcal{O}^{\emptyset}$ be the set of closed objects.*

(i) *The top-level rules are*

$$
\begin{array}{lll}
(\beta\text{–Obj}) & (\lambda x{:}A.M)\, N \to_{\beta} M[N/x] & (\lambda x_{\emptyset}{:}A.M)\, N \to_{\beta} M[N/x] \text{ if } N \in \mathcal{O}^{\emptyset} \\
(\beta\text{–Fam}) & (\lambda x{:}A.B)\, N \to_{\beta} B[N/x] & (\lambda x_{\emptyset}{:}A.B)\, N \to_{\beta} B[N/x] \text{ if } N \in \mathcal{O}^{\emptyset} \\
(\beta\text{–Kinds}) & (\lambda x{:}A.K)\, N \to_{\beta} K[N/x] & (\lambda x_{\emptyset}{:}A.K)\, N \to_{\beta} K[N/x] \text{ if } N \in \mathcal{O}^{\emptyset}
\end{array}
$$

(ii) *one-step, many-steps reduction and congruence are defined as usual.*

The two notions of β-reduction in CLF, namely standard β-reduction and restricted β-closed reduction, nicely combine, in the sense that a potential β-closed reduction is preserved under application of any substitution (coming from another, possibly standard reduction).

Lemma 4.3 (Closure under Reduction and Substitution) *If $N \in \mathcal{O}^{\emptyset}$, then, for any substitution θ, $N\theta \in \mathcal{O}^{\emptyset}$. Moreover, for any N and T, and for any θ such that $x \notin \mathsf{CoDom}(\theta)$, we have $T[N/x]\theta \equiv T\theta[N\theta/x]$.*

Using the above lemma, one can prove the following substitution lemma.

Lemma 4.4 (Substitution) *If $T \mapsto\!\!\!\!\to_{\beta} T'$ and $\theta \mapsto\!\!\!\!\to_{\beta} \theta'$, then $T\theta \mapsto\!\!\!\!\to_{\beta} T'\theta'$.*

Using Lemma 4.3 and the Substitution Lemma above, and following the standard argument based on parallel reduction, one can prove:

Theorem 4.5 (Confluence) *The relation $\mapsto_{\beta}$ is confluent.* □

Signatures rules

$$\frac{}{\emptyset \text{ sig}} \ (S{\cdot}Empty)$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma K \quad a \notin \mathsf{Dom}(\Sigma)}{\Sigma, a{:}K \text{ sig}} \ (S{\cdot}Kind)$$

$$\frac{\Sigma \text{ sig} \quad \vdash_\Sigma A : \mathsf{Type} \ c \notin \mathsf{Dom}(\Sigma)}{\Sigma, f{:}A} \ (S{\cdot}Type)$$

Contexts rules

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \emptyset} \ (C{\cdot}Empty)$$

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

Kind rules

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

$$\frac{\Gamma, \overline{x}{:}A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi\overline{x}{:}A.K} \ (K{\cdot}Pi)$$

$$\frac{\Gamma, \overline{x}{:}A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \lambda\overline{x}{:}A.K} \ (K{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma \Pi\overline{x}{:}A.K \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma (\lambda\overline{x}{:}A.K)\, N} \ (K{\cdot}Appl)$$

Families rules

$$\frac{\vdash_\Sigma \Gamma \quad a{:}K \in \Gamma}{\Gamma \vdash_\Sigma a : K} \ (F{\cdot}Var)$$

$$\frac{\Gamma, \overline{x}{:}B \vdash_\Sigma A : \mathsf{Type}}{\Gamma \vdash_\Sigma \Pi\overline{x}{:}B.A : \mathsf{Type}} \ (F{\cdot}Pi)$$

$$\frac{\Gamma, \overline{x}{:}B \vdash_\Sigma A : K}{\Gamma \vdash_\Sigma \lambda\overline{x}{:}B.A : \Pi\overline{x}{:}B.K} \ (F{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma A : \Pi\overline{x}{:}B.K \quad \Gamma \vdash_\Sigma N : B}{\Gamma \vdash_\Sigma A\, N : (\lambda\overline{x}{:}B.K)\, N} \ (F{\cdot}Appl)$$

$$\frac{\Gamma \vdash_\Sigma A : K' \quad \Gamma \vdash_\Sigma K \qquad \Gamma \vdash_\Sigma K =_\beta K'}{\Gamma \vdash_\Sigma A : K} \ (F{\cdot}Conv)$$

Object rules

$$\frac{\vdash_\Sigma \Gamma \quad \overline{x}{:}A \in \Gamma}{\Gamma \vdash_\Sigma \overline{x} : A} \ (O{\cdot}Var)$$

$$\frac{\vdash_\Sigma \Gamma \quad f{:}A \in \Sigma}{\Gamma \vdash_\Sigma f : A} \ (O{\cdot}Const)$$

$$\frac{\Gamma, \overline{x}{:}B \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma \lambda\overline{x}{:}B.M : \Pi\overline{x}{:}B.A} \ (O{\cdot}Abs)$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi\overline{x}{:}B.A \quad \Gamma \vdash_\Sigma N : B}{\Gamma \vdash_\Sigma M\, N : (\lambda\overline{x}{:}B.A)\, N} \ (O{\cdot}Appl)$$

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

Figure 4. CLF Type System

4.3 CLF *Type System*

CLF involves classical type judgments of the following shape:

$$\Sigma \text{ sig} \qquad \vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma A : \mathsf{Type} \qquad \Gamma \vdash_\Sigma M : A$$

The typing rules of CLF are given in Figure 4. As was the case for PLF, we have also here the auxiliary equality judgments $\Gamma \vdash_\Sigma M =_\beta N$. As for PLF, confluence holds for raw terms, hence equality judgments are unproblematic. Due to the simplicity of predicates, the metatheory of CLF follows from that of LF [19], with minor modifications. The following *gallery* of results holds:

Proposition 4.6 (Gallery) (i) *Subderivation Property;*

(ii) *Derivability of Permutation and Weakening;*

(iii) *Unicity of Types and Kinds;*

(iv) *Transitivity;*

(v) *Abstraction Typing;*

(vi) *Subject Reduction.*

Strong Normalization follows from strong normalization of standard LF, observing that β-reduction restricted to closed arguments is a special case of the plain β-reduction.

Theorem 4.7 (Strong Normalization for CLF) (i) *If* $\Gamma \vdash_\Sigma K$, *then* $K \in \mathsf{SN}^{\mathcal{K}}$;

(ii) *If* $\Gamma \vdash_\Sigma A : K$, *then* $A \in \mathsf{SN}^{\mathcal{F}}$;

(iii) *If* $\Gamma \vdash_\Sigma M : A$, *then* $M \in \mathsf{SN}^{\mathcal{O}}$. □

5 Putting GLF to use: Examples

In this section we illustrate by means of a few simple examples how PLF and CLF can be conveniently used as Logical Frameworks. Clearly, more experiments are necessary in order to assess in full generality the potential of such Frameworks. But we are confident that already these very simple encodings of logical systems, which are problematic in standard LF, make the point concerning the usability of the new Frameworks. Further possible developments will be mentioned in Section 6. We assume the reader familiar with the pragmatics of Logical Frameworks. An elementary introduction appears in [1]. Most of the papers cited in the Introduction provide further interesting material.

5.1 Case Analysis in PLF

Case analysis can be handled very easily and neatly in PLF by taking advantage of the pattern matching facilities. For instance, in order to encode in PLF the predecessor, for the classical (untyped) term rewriting system over the constant integer type int, $(0 \to 0, (succ\ x) \to x)$, we can simply write $\lambda 0{:}\mathsf{int}.0$ and $\lambda(succ\ x){:}[x{:}\mathsf{int}].x$. Following van Oostroom [31], and [4], we can take advantage of having functions-as-patterns. Namely, projections for pairs can be neatly defined as follows.

$$\mathsf{Pi}_1 \triangleq \lambda(\lambda z{:}\mathsf{bool}.z\ x\ y){:}[x{:}A, y{:}B].x \qquad \mathsf{Pi}_2 \triangleq \lambda(\lambda z{:}\mathsf{bool}.z\ x\ y){:}[x{:}A, y{:}B].y$$

where bool is the constant boolean type.

5.2 Plotkin's Call-by-value Lambda Calculus.

For lack of space, we will provide only one example encoding to illustrate how patterns can increase the usability of Dependent Type Theory as a metalanguage

for encoding logical systems. Another encoding appears in [25]. Plotkin's call-by-value lambda calculus (λ_{v}-calculus) [35] differs from the traditional lambda calculus in the formulation of the β_{v}-reduction rule, namely $(\lambda x.M)\,N \to_{\beta_{\mathsf{v}}} M[N/x]$ provided that N is a value, that is a variable or an abstraction. The η-reduction rule is the usual $(\lambda x.M\,x) \to_{\eta} M$, provided $x \notin \mathsf{Fv}(M)$, since variables are intended to range over values. Although interesting encodings of Plotkin's λ_{v}-calculus do exist in standard LF, the price to pay is to introduce an auxiliary machinery for representing syntactic subcategories, [1]. In PLF we can present alternate encodings of Plotkin's λ_{v}-calculus which safely do away with subcategories, as in the signature appearing in Figure 5. In the signature Σ_{v} of Figure 5 standard abbreviations are in use, *i.e.*

Syntactic Categories

o : Type

Constructors and Judgments

$! : o^2$ $\quad \mathsf{Lam} : \Pi f{:}[\Pi !\, x^o.o].\, o$ $\quad \mathsf{App} : o^3$ $\quad = : o \to o \to \mathsf{Type}$

Axioms and Rules

$\mathsf{Eq}_{\mathsf{refl}}$: $\Pi x^o.\, x = x$

$\mathsf{Eq}_{\mathsf{symm}}$: $\Pi x^o.\, \Pi y^o.\, (x = y) \to (y = x)$

$\mathsf{Eq}_{\mathsf{trans}}$: $\Pi x^o.\, \Pi y^o.\, \Pi z^o.\, (x = y) \to (y = z) \to (x = z)$

$\mathsf{Eq}_{\mathsf{ctx}}$: $\Pi x^o.\, \Pi y^o.\, \Pi z^o.\, \Pi w^o.\, (x = y) \to (z = w) \to (\mathsf{App}\; x\, z = \mathsf{App}\; y\, w)$

Betav : $\Pi f{:}[\Pi !\, x^o.o].\, \Pi y^o.\, \mathsf{App}\; (!\,(\mathsf{Lam}\; f))\,(!\, y) = f\,(!\, y)$

Xiv : $\Pi f{:}[\Pi !\, x^o.o].\, \Pi g{:}[\Pi !\, x^o.o].$

$\qquad (\Pi z^o.\; f\,(!\, z) = g\,(!\, z) \to (!\,(\mathsf{Lam}\; f) = !\,(\mathsf{Lam}\; g))$

Etav : $\Pi x^o.\, !\,(\mathsf{Lam}\; (\lambda(!\, y^o).\mathsf{App}\; (!\, x)\,(!\, y))) = !\, x$

Figure 5. The signature Σ_{v} for Plotkin's λ_{v}-calculus in PLF

infix notation, operators precedence, $\Pi x{:}A.B \equiv A \to B$, if $x \notin FV(B)$, as well as the following ones: o^n for $\overbrace{o \to \ldots \to o}^{n \text{ times}}$ and $\checkmark\mathsf{C}[x]{:}[x{:}o]$ for $\checkmark\mathsf{C}[x^o]$.

All the constants are self-explicatory but for ! This constructor denotes values, and coherently, the domain of the Lam constructor takes as arguments only functions whose argument has to have the pattern of a value. Please notice the essential use of patterns. The rationale of this signature is clarified by the following adequacy theorem:

Theorem 5.1 (Adequacy and Faithfulness) *Let $\Xi_\Gamma(o)$ be the set of* PLF *terms in normal form of type o in the context $\Gamma \equiv [x_1{:}o, \ldots, x_n{:}o]$, and let*

Propositional Connectives and Judgment

o : Type $\quad \supset\colon o^3 \quad \neg : o^2 \quad \Box : o^2 \quad \mathsf{True} : o \to \mathsf{Type}$

Propositional Axioms

$\mathsf{A}_1 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset (\psi \supset \phi)$

$\mathsf{A}_2 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \Pi\theta^o.\ \mathsf{True}(\phi \supset (\psi \supset \theta)) \supset (\phi \supset \psi) \supset (\phi \supset \theta)$

$\mathsf{A}_3 \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}(\neg\psi \supset \neg\phi) \supset ((\neg\psi \supset \phi) \supset \psi$

Modal Axioms

$\mathsf{K} \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\Box(\phi \supset \psi) \supset (\neg\phi \supset \neg\psi)$

$4 \quad : \Pi\phi^o.\ \mathsf{True}\Box\phi \supset \Box\Box\phi$

$\top \quad : \Pi\phi^o.\ \mathsf{True}\Box\phi \supset \phi$

Rules

$\mathsf{MP} \quad : \Pi\phi^o.\ \Pi\psi^o.\ \mathsf{True}\phi \supset \mathsf{True}\phi \supset \psi \to \mathsf{True}\psi$

$\mathsf{NEC} : \Pi\phi^o.\ \Pi x_\emptyset{:}\mathsf{True}\phi.\ \mathsf{True}\Box\phi$

Figure 6. The signature Σ_{S_4} for classic S_4 modal logic in Hilbert style in CLF

a $[\![-]\!]_\Gamma : \Lambda_{\mathsf{v}}[x_1, \ldots, x_n] \longrightarrow \Xi_\Gamma(o)$ *be the bijective function defined as follows.*

$$[\![M]\!]_\Gamma = \begin{cases} !\, x & \text{if } M \equiv x \\ \mathsf{App}[\![P]\!]_\Gamma\, [\![Q]\!]_\Gamma & \text{if } M \equiv P\,Q \\ !\,(\mathsf{Lam}\ (\lambda!\, x^o.[\![P[x]]\!]_{\Gamma, x:o})) & \text{if } M \equiv \lambda x.P[x] \end{cases}$$

and let $\vdash_{\mathsf{v}} M = N$ *denote the standard equational theory for Plotkin's* λ_{v}*-calculus [35]. The following holds:*

(i) $\Gamma \vdash_{\Sigma_{\mathsf{v}}} [\![M]\!]_\Gamma : o$ *is provable if and only if* $M \in \Lambda_{\mathsf{v}}[x_1, \ldots, x_n]$ *(i.e. the set of terms in* Λ_{v} *with* $x_1, \ldots, x_n$ *free variables).*

(ii) $\Delta \vdash_{\Sigma_{\mathsf{v}}} P : [\![M]\!]_\Gamma = [\![N]\!]_\Gamma$ *is provable, for* $\Delta \triangleq y_1{:}[\![M_1]\!]_\Gamma = [\![N_1]\!]_\Gamma, \ldots,$ $y_n{:}[\![M_n]\!]_\Gamma = [\![N_n]\!]_\Gamma$ *and some* P*, iff* $M_1 = N_1, \ldots, M_n = N_n \vdash_{\mathsf{v}} M = N$*.*

5.3 Modal Logics

The expressive power of the Closed Logical Framework allows to encode smoothly *rules of proof*, *i.e.* rules which apply only to premises which do not depend on any assumption, such as the rule of *necessitation* in Modal Logic, as well as *rules of derivation*, such as *modus ponens*. It uses a constrained Π-abstraction in rules of

proof and a standard Π-abstraction in rules of derivation.

We shall not develop here the encodings of all the plethora of modal logics, in Hilbert and Natural Deduction style, which appear in [2]. By way of example, we shall only give the signature for classical S_4 in Hilbert style, which features necessitation as a rule of proof, namely

$$\frac{\emptyset \vdash \phi}{\emptyset \vdash \Box\phi}\ (NEC)$$

The predicate $\mathrm{Closed}_x \triangleq$ "x is a term with no free variables" is precisely what is needed to encode it correctly.

The signature Σ_{S_4} encoding the modal logic S_4 in CLF is presented in Figure 6. Standard abbreviations are in use. Notice that, apart from the encoding of the rule of proof NEC, all the remaining constants are standard. We can easily show that:

Theorem 5.2 (Logical Adequacy) *$\phi_1, \ldots, \phi_n \vdash_{S_4} \psi$ if and only if $\exists M.\ \Gamma, \mathsf{True}\phi_1, \ldots, \mathsf{True}\phi_n \vdash_{\Sigma_{S_4}} M : \mathsf{True}\psi$, where $\Gamma \equiv X_1{:}o, \ldots, X_k{:}o$ for X_i free propositional variables in $\phi_1, \ldots, \phi_n, \psi$.*

Adequacy of proof encodings in CLF is usually straightforward. On the other hand, when explicit encodings of the closure judgment are given in LF, to achieve adequacy one needs to prove that there exists at most one derivation of such a judgment.

6 Conclusions and Directions for Future Work

In this paper, we have introduced a general Logical Framework which subsumes the Logical Framework LFof [19], and generates new Logical Frameworks. These can feature a very broad spectrum of generalized β-reductions, together with an expressive type system which records when such reductions do not fire. The key ingredient in the typing system is a decomposition of the standard term-application rule.

We have instantiated our Framework to two important case-studies. The Pattern Lambda Calculus PLF, which arises from the tradition of [31, 9, 10, 4], and the Closed Logical Framework CLF. For both calculi we have have studied in depth the language theory, proving major metatheoretical results, such as subject reduction, confluence, strong normalization. In both cases we achieve decidability, which legitimates them as metalanguages for proof checking and interactive proof editing. Finally, we have illustrated the usability and expressivity of such Frameworks giving some examples of encodings which where hitherto problematic in standard LF. We believe that our metalogical Framework has some considerable potential, but more experiments need to be done to show this. A thorough comparison with existing work is also mandatory. Among various results, we prove also strong normalization via reducibility candidates, for a pattern lambda calculus PLF. This problem was left open in [4], already for a weaker subsystem. A strong normalization proof for a weaker system than PLF appears in [39]. Here is a rather rhapsodic list of comments and directions for future work.

- Formalize the notion of predicate $\mathcal{P}$, still preserving generality.
- We conjecture that confluence and strong normalization properties can be established for a generic predicate calculus, provided that the various notions of reductions nicely combine, in the sense that $\mapsto_{\mathcal{P}_i}$-reductions are preserved both under $\mapsto_{\mathcal{P}_j}$-reductions of the argument and application to the argument of any substitution coming from other reductions.
- Case analysis in PLF should be compared with that of inductive types in Coq.
- Instantiate GLF so as to provide a more natural encoding of the Natural Deduction $\Box$-*introduction rule* of Prawitz:

$$\frac{\Box\Gamma \vdash \phi}{\Box\Gamma \vdash \Box\phi}\ (\Box\text{-}Intro)$$

E.g. if we introduce a new predicate $\mathrm{Occurs}_x \triangleq$ "x is a term whose free variables occur only in subterms of type $\mathsf{True}\Box\psi$ for some ψ", then $\Box$-$Intro$ becomes:

$$\Box\text{-}I : \Pi\phi{:}o.\ \Pi\mathrm{Occurs}_x{:}[x{:}\mathsf{True}\phi].\ \mathsf{True}\Box\phi.$$

- Section 3.3 shows that there is no strong notion of pattern reduction. Still, can we allow reductions in patterns under specific strategies, *e.g.* only where the pattern is in normal form according to Definition 3.9 and does not contain K-redexes?
- Can the linearity restriction in patterns be relaxed, still preserving confluence and strong normalization over well typed patterns?
- Our results should scale up to all the systems in [4], *i.e.* to systems corresponding to the full Calculus of Constructions [7].
- Is there an interesting Curry-Howard isomorphism for PLF and more generally for systems blending rewriting facilities and higher order calculi?
- Instantiate GLF in order to give sharp encodings of relevance and linear logics?
- Extend existing proof assistants based on dependent type systems, *e.g.* Coq, with pattern matching facilities as in PLF, and more generally with GLF.
- Among the various calculi with patterns, versions *à la* Curry of PLF should be explored and compared e.g. with the pattern calculus recently introduced in [21].

Acknowledgement

The last author would like to thank warmly Horatiu Cirstea and Claude Kirchner for many useful discussions, and for infecting him with the intellectual virus of blending term rewriting with lambda calculus. Special thanks to BenjaMin Wack (a.k.a. BMW) for showing an interesting non-normalizing term in an early version of PLF, and for many useful discussions. Also thanks to Philippe Audebaud for a careful reading of the paper. Finally, the authors thank the anonymous referee for useful comments.

References

[1] A. Avron, F. Honsell, I.A. Mason, and R. Pollack. Using Typed Lambda Calculus to Implement Formal Systems on a Machine. *Journal of Automated Reasoning*, 9(3):309–354, 1992.

[2] A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding Modal Logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, 1998.

[3] Alf. The ALF Home Page, 2006. http://www.cs.chalmers.se/~sydow/alf/all.html.

[4] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Pattern Type Systems. In *Proc. of POPL*, pages 250–261. The ACM Press, 2003.

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

[6] M. Cresswell and G. Hughes. *A companion to Modal Logic*. Methuen, 1984.

[7] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[8] R. Constable and D. Howe. NuPRL as a General Logic. In P. Odifreddi, editor, *Logic and Computation*. Academic Press, 1990.

[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] R. L. Constable, *et. al.* *Implementing Mathematics with the Nuprl Proof Development System*. Prentice–Hall, Englewood Cliffs, NJ, 1986.

[12] Coq. The Coq Home Page, 2006. http://coq.inria.fr.

[13] T. Coquand, R. Pollack, and M. Takeyama. A Logical Framework with Dependently Typed Records. In *Prof. of TLCA*, volume 2701 of *LNCS*, pages 105–119. Springer-Verlag, 2003.

[14] N. G. de Bruijn. A Survey of the Project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.

[15] J. Despeyroux. A Higher-Order Specification of the π-calculus. In *Proc. of IFIP TCS*, volume 1872 of *LNCS*, pages 425–439. Springer-Verlag, 2000.

[16] G. Dowek, T. Hardin, and C. Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.

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

[18] S. Feferman. Finitary Inductively Presented Logics. In *Proc. of Logic Colloquium*. Wiley, 1988.

[19] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *Journal of the ACM*, 40(1):143–184, 1993. Preliminary version in proc. of LICS'87.

[20] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (Co)Inductive Type Theories. *Theoretical Computer Science*, 253(2):239–285, 2001.

[21] B. Jay and D. Kesner. Pure pattern calculus. In *Proc. of the European Symposium on Programming (ESOP)*, volume 3924 of *LNCS*, pages 100–114. Springer-Verlag, 2006.

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

[23] F. Kamareddine, R. Bloo, and R. Nederpelt. On π-conversion on the λ-cube and the Combination with Abbreviations. *Annals of Pure and Applied Logics*, 97(1-3):27–45, 1999.

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

[25] L. Liquori, F. Honsell, and R. Redamalla. A Language for Verification and Manipulation of Web Documents. In *Proc. of WWV*, pages 127–137. Electronic Notes in Theoretical Computer Science, 2005.

[26] Z. Luo. ECC: An Extended Calculus of Constructions. In *Proc. of LICS*, pages 385–395, 1990.

[27] P. Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984.

[28] P. Martin-Lof. Truth of a Proposition, Evidence of a Judgement, Validity of a Proof. Lecture at the Workshop Theories of Meaning, Florence, Italy, 1985.

[29] R. Nederpelt, J. H. Geuvers, and R. de Vrijer (eds.). *Selected Papers on Automath*. North-Holland, 1994.

[30] 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.

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

[32] V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, 1994.

[33] L. Paulson. Interactive Theorem Proving with Cambridge LCF. Technical Report 80, Computer Laboratory, University of Cambridge, 1985.

[34] F. Pfenning. Logical Frameworks. In *Handbook of Automated Reasoning*, volume II, pages 1063–1147. Elsevier and MIT Press, 2001.

[35] G. Plotkin. Call by Name, Call by Value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[36] F. Pfenning and C. Schurmann. System description: Twelf – a Meta-logical Framework for Deductive Systems. In *Proc. of CADE*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 202–206, 1999.

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

[38] TYPES-WG. In *Proc. of Types*, volume 806, 996, 1158, 1512, 1657, 2277, 2646, 3085, 3839 of *LNCS*. Springer-Verlag.

[39] B. Wack. *Typage et déduction dans le calcul de réécriture*. PhD thesis, Université Henri Poincaré - Nancy I, 2005.

A Appendix

A.1 Proof of Lemma 3.11

(i) By induction on P.
- $P \equiv f$ or $P \equiv x$, then the thesis is immediate.
- $P \equiv f\,P_1\,\ldots\,P_n$. Then, $N \equiv f\,N_1\,\ldots\,N_n$, with $\theta_i = \mathcal{A}lg(P_i;N_i)$, $\theta = \bigcup_i \theta_i$, and $N_i \mapsto^0_\beta N'_i$, for all i (and, for exactly one i, $N_i \mapsto_\beta N'_i$). By induction hypothesis, for all i, there exists θ'_i, such that $\theta'_i = \mathcal{A}lg(P_i;N'_i)$, and $\theta_i \mapsto\!\!\!\!\to_\beta \theta'_i$. Now, by the linearity hypothesis on P, the θ'_i's are all coherent, thus we can define $\theta' \stackrel{\triangle}{=} \bigcup_i \theta'_i$, such that $\theta' = \mathcal{A}lg(f\,P_1\,\ldots\,P_n, f\,N'_1\,\ldots\,N'_n)$.
- $P \equiv \lambda P_1{:}\Delta.P_2$. Then, $N \equiv \lambda P_1{:}\Delta.N_2$, with $\theta = \mathcal{A}lg(P;N) = \mathcal{A}lg(P_2;N_2)$, $\theta(x) = x$, for all $x \in \mathsf{Fv}(P_1)$, and $\lambda P_1{:}\Delta.N_2 \mapsto_\beta \lambda P_1{:}\Delta.N'_2 \equiv N'$. By induction hypothesis, there exists θ', such that $\theta' = \mathcal{A}lg(P_2;N'_2)$ and $\theta \mapsto\!\!\!\!\to_\beta \theta'$, hence $\theta' = \mathcal{A}lg(P;N')$.
- $P \equiv (\lambda P_1{:}\Delta.P_2)\,P_3\,\boldsymbol{P}$, with $\mathcal{A}lg(P_1;P_3\overline{\theta}) = \mathsf{fail}$, for all $\overline{\theta}$. Then, $N \equiv (\lambda P_1{:}\Delta.N_2)\;N_3\,\boldsymbol{N}$, and $\mathcal{A}lg(P_1;N_3) = \mathsf{fail}$, and $\theta = \theta_1 \cup \theta_2 \cup \boldsymbol{\theta}$, where $\theta_1 = \mathcal{A}lg(P_2;N_2)$, and $\theta_2 = \mathcal{A}lg(P_3;N_3)$, and $\boldsymbol{\theta} = \mathcal{A}lg(\boldsymbol{P};\boldsymbol{N})$ ($\theta_1, \theta_2, \boldsymbol{\theta}$ are the identity on the free variables of P_1), and $(\lambda P_1{:}\Delta.N_2)\,N_3\,\boldsymbol{N} \mapsto_\beta (\lambda P_1{:}\Delta.N'_2)\,N'_3\,\boldsymbol{N'} \equiv N'$. By induction hypothesis, there exist $\theta'_1 = \mathcal{A}lg(P_2,N'_2)$, and $\theta'_2 = \mathcal{A}lg(P_3,N'_3)$, and $\boldsymbol{\theta'} = \mathcal{A}lg(\boldsymbol{P},\boldsymbol{N'})$ such that $\theta_1 \mapsto\!\!\!\!\to_\beta \theta'_1$, and $\theta_2 \mapsto\!\!\!\!\to_\beta \theta'_2$, and $\boldsymbol{\theta} \mapsto\!\!\!\!\to_\beta \boldsymbol{\theta'}$. By the linearity hypothesis on P, the θ'_i's are all coherent, thus we have $\theta' \stackrel{\triangle}{=} \bigcup_i \theta'_i = \mathcal{A}lg(P;N')$.

(ii) We proceed by induction on P.
- $P \equiv f$. Then, the thesis is immediate.
- $P \equiv x$, and $\overline{\theta} \equiv [N/x]$, and $\overline{\theta}' \equiv [N\theta/x]$. Then, the thesis follows by proving, by induction on T, that, if θ does not overlap with $[N/x]$, then $T[N/x]\theta \equiv T\theta[N\theta/x]$.
- $P \equiv f\,P_1\,\ldots\,P_n$. Then, $N \equiv f\,N_1\,\ldots\,N_n$, and $\overline{\theta} = \bigcup_i \overline{\theta}_i$, and $\theta_i = \mathcal{A}lg(P_i;N_i)$, for all i. By induction hypothesis, for all i, there exists $\theta'_i = \mathcal{A}lg(P'_i;N'_i)$, such that, for all T, we have $T\overline{\theta}_i\theta = T\theta\overline{\theta}'_i$. Then, the thesis follows by the fact that the $\overline{\theta}_i$'s ($\overline{\theta}'_i$'s) are all coherent, since patterns satisfy the linearity condition on variables.
- $P \equiv \lambda P_1{:}\Delta.P_2$, and $\overline{\theta} = \mathcal{A}lg(P;N)$. Then, $\overline{\theta}(x) = x$, for all $x \in \mathsf{Fv}(P_1)$, $N \equiv \lambda P_1{:}\Delta.N_2$, and

$$\frac{}{T \Rightarrow_\beta T}\ (Par_1) \qquad \frac{T \Rightarrow_\beta T' \quad N \Rightarrow_\beta N'}{T\,N \Rightarrow_\beta T'\,N'}\ (Par_2)$$

$$\frac{T \Rightarrow_\beta T' \quad \Delta \Rightarrow_\beta \Delta'}{\checkmark P{:}\Delta.T \Rightarrow_\beta \checkmark P{:}\Delta'.T'}\ (Par_3) \qquad \frac{\Delta \Rightarrow_\beta \Delta' \quad T \Rightarrow_\beta T' \quad N \Rightarrow_\beta N' \quad \mathcal{A}lg(P;N') = \theta}{(\lambda P{:}\Delta.T)\,N \Rightarrow_\beta T'\theta}\ (Par_4)$$

Figure A.1. Parallel Reduction

$\overline{\theta} = \mathcal{A}lg(P_2; N_2)$. By induction hypothesis, for any θ non-overlapping with $\overline{\theta}$ (in particular $\overline{\theta}$ does not overlap with the bound variables of P), there exists $\overline{\theta}'$, such that $\overline{\theta}' = \mathcal{A}lg(P_2, N_2\theta)$, and, for all T, we have $T\overline{\theta}\theta \equiv T\theta\overline{\theta}'$. Hence, we also have $\overline{\theta}' = \mathcal{A}lg(\lambda P_1{:}\Delta.P_2, \lambda P_1{:}\Delta\theta.N_2\theta)$.
• $P \equiv (\lambda P_1{:}\Delta.P_2)\ P_3\ \boldsymbol{P}$, and $\overline{\theta} = \mathcal{A}lg(P; N)$. Then, $N \equiv (\lambda P_1{:}\Delta.N_2)\ N_3\ \boldsymbol{N}$, with $\overline{\theta} = \overline{\theta_1} \cup \overline{\theta_2} \cup \boldsymbol{\theta}$, where $\overline{\theta_1} = \mathcal{A}lg(P_2; N_2)$, and $\overline{\theta_2} = \mathcal{A}lg(P_3; N_3)$, and $\boldsymbol{\theta} = \mathcal{A}lg(\boldsymbol{P}; \boldsymbol{N})$ (and $\theta_1, \theta_2, \boldsymbol{\theta}$ are the identity on the free variables of P_1). By induction hypothesis, for any non-overlapping θ, there exist $\overline{\theta}'_1 = \mathcal{A}lg(P_2, N_2\theta)$, and $\overline{\theta}'_2 = \mathcal{A}lg(P_3, N_3\theta)$, $\boldsymbol{\theta'} = \mathcal{A}lg(\boldsymbol{P}, \boldsymbol{N'})$, such that, for all T, we have $T\overline{\theta}\theta \equiv T\theta\overline{\theta}'$. By the linearity hypothesis on P, the θ'_i's are all coherent, thus we have $\theta' \triangleq \bigcup_i \theta'_i = \mathcal{A}lg(P; N')$. □

A.2 Proof of Confluence Theorem 3.13.

Definition A.1 (Parallel Reduction) *The parallel reduction $\Rightarrow_\beta$ is defined in Figure A.1.*

It is easy to prove that:

Lemma A.2 (Relations) $\mapsto_\beta \subseteq \Rightarrow_\beta \subseteq \twoheadrightarrow_\beta$.

By Lemma A.2 above, in order to prove the confluence of the $\mapsto_\beta$ relation, it is enough to prove the diamond property of the parallel reduction $\Rightarrow_\beta$. To this aim, we need the following mapping $\diamond$, and a number of instrumental lemmas.

Definition A.3 (Diamond) *We define $\diamond$ by induction (point-wise extended to contexts):*

$$\begin{aligned}
x^\diamond &\triangleq x \\
(\checkmark P{:}\Delta.T)^\diamond &\triangleq \checkmark P{:}\Delta^\diamond.T^\diamond \\
(T\,N)^\diamond &\triangleq T^\diamond\,N^\diamond && \text{if } T \text{ is not an abstraction} \\
((\lambda P{:}\Delta.T)\,N)^\diamond &\triangleq \begin{cases} T^\diamond\theta & \text{if } \mathcal{A}lg(P; N^\diamond) = \theta \\ (\lambda P{:}\Delta.T)^\diamond\,N^\diamond & \text{otherwise} \end{cases}
\end{aligned}$$

Lemma A.4 *For any T, we have $T \Rightarrow_\beta T^\diamond$.*

The following lemma is the counterpart of Lemma 3.11(i) for $\Rightarrow_\beta$, and it expresses the fact that matchings are preserved under $\Rightarrow_\beta$-reductions.

Lemma A.5 (Parallel Reduction Preserves Matching) *If $\theta = \mathcal{A}lg(P; N)$ and $N \Rightarrow_\beta N'$, then there exists θ', such that $\theta' = \mathcal{A}lg(P; N')$ and $\theta \Rightarrow_\beta \theta'$.*

Lemma A.6 (Parallel Substitution) *If $T \Rightarrow_\beta T'$ and $\theta \Rightarrow_\beta \theta'$, then $T\theta \Rightarrow_\beta T'\theta'$.*

Proof *By induction on the derivation of $T \Rightarrow_\beta T'$. If $T \Rightarrow_\beta T$ is obtained by an application of rule* (Par_1)*, then the thesis follows by proving that, if $\theta \Rightarrow_\beta \theta'$, then for all T, $T\theta \Rightarrow_\beta T\theta'$ (which can be shown by straightforward induction on T). The remaining cases are dealt with straightforwardly using the induction hypothesis, except for the case where the last rule applied in the derivation is* (Par_4)*, i.e.:*

$$\frac{\Delta \Rightarrow_\beta \Delta' \quad T_1 \Rightarrow_\beta T'_1 \quad N \Rightarrow_\beta N' \quad \mathcal{A}lg(P; N') = \overline{\theta}}{T \equiv (\lambda P{:}\Delta.T_1)\,N \Rightarrow_\beta T'_1\overline{\theta} \equiv T'}\ (Par_4)$$

By induction hypothesis, $\Delta\theta \Rightarrow_\beta \Delta'\theta'$, and $T_1\theta \Rightarrow_\beta T'_1\theta'$, and $N\theta \Rightarrow_\beta N'\theta'$. Moreover, by Lemma 3.11(ii), there exists $\overline{\theta}' = \mathcal{A}lg(P; N'\theta')$. Thus, by rule ($\mathsf{Par}_4$)*, we have $(\lambda P{:}\Delta\theta.T_1\theta)\,N\theta \Rightarrow_\beta T'_1\theta'\overline{\theta}' \equiv T'_1\overline{\theta}\theta'$, by Lemma 3.11(ii). This concludes the proof.* □

Lemma A.7 (Diamond Property) *If $T_1 \Rightarrow_\beta T_2$, then $T_2 \Rightarrow_\beta T_1^\diamond$.*

Proof *By induction on the derivation of $T_1 \Rightarrow_\beta T_2$. If the only rule applied in the derivation is* (Par_1)*, then the thesis follows by Lemma A.4. If the last rule in the derivation is* (Par_2) *or* (Par_3)*, then the thesis follows by induction hypothesis. Finally, let us consider the case when the last rule in the derivation is* (Par_4)*, i.e.:*

$$\frac{\Delta \Rightarrow_\beta \Delta' \quad T \Rightarrow_\beta T' \quad N \Rightarrow_\beta N' \quad \mathcal{A}lg(P; N') = \theta}{T_1 \equiv (\lambda P{:}\Delta.T)\,N \Rightarrow_\beta T'\theta \equiv T_2}\ (Par_4)$$

By induction hypothesis, $\Delta' \Rightarrow_\beta \Delta^\diamond$, and $T' \Rightarrow_\beta T^\diamond$, and $N' \Rightarrow_\beta N^\diamond$. Hence, by Lemma A.5, there exists $\theta' = \mathcal{A}lg(P; N^\diamond)$, and $\theta \Rightarrow_\beta \theta'$. Thus, by definition of $(\ )^\diamond$, we have $T_1^\diamond \equiv T^\diamond\theta'$, and, by the Substitution Lemma, $T'\theta \Rightarrow_\beta T^\diamond\theta'$. □

Finally, Theorem 3.13 follows by Lemmas A.2 and A.7.

A.3 Proof of Subject Reduction Theorem 3.20

Proof of Lemma Abstraction Typing 3.19 By induction on derivations. □
The proof of Subject Reduction Theorem 3.20 follows by induction on the structure of derivations, using Abstraction Typing and Transitivity. □

A.4 Proof of Strong Normalization Theorem 3.21.

Proof of Lemma 3.23. We only prove that $\mathsf{SN}^{\mathcal{O}} \in \mathsf{Comp}^{\mathcal{O}}$. The proof of $\mathsf{SN}^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{F}}$ being similar. The set $\mathsf{SN}^{\mathcal{O}}$ clearly satisfies (**c1**) and (**c2**). We prove that $\mathsf{SN}^{\mathcal{O}}$ satisfies property (**c3**). Let assume that $Q \mapsto\!\!\!\!\rightarrow_\beta Q'$, and $\mathcal{A}lg(P; Q') = \theta$, and $(M\theta)\boldsymbol{N} \in \mathsf{SN}^{\mathcal{O}}$, and $\mathsf{CoDom}(\Delta)$, and $Q \in \mathsf{SN}$. We have to prove that $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N} \in \mathsf{SN}^{\mathcal{O}}$. We proceed by induction on the lengths of the minimal derivations to normal forms of $(M, Q, \boldsymbol{N}, \mathsf{CoDom}(\Delta))$, lexicographically ordered. If $M, Q, \boldsymbol{N}, \mathsf{CoDom}(\Delta)$ are all normal forms, then the thesis is immediate. Otherwise, let us consider all possible $\mapsto_\beta$-reductions starting from $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N}$. We have to prove that the reduced terms are strongly normalizing. There are various cases:

- $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N} \mapsto_\beta (\lambda P{:}\Delta.M')\, Q\, \boldsymbol{N}$. Since $M\theta \in \mathsf{SN}^{\mathcal{O}}$, by hypothesis, and $M\theta \mapsto\!\!\!\!\rightarrow_\beta M'\theta$, by Lemma 3.12, then $M'\theta \in \mathsf{SN}^{\mathcal{O}}$. Thus, by induction hypothesis, $(\lambda P{:}\Delta.M')\, Q\, \boldsymbol{N} \in \mathsf{SN}^{\mathcal{O}}$;
- $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N} \mapsto_\beta (\lambda P{:}\Delta.M)\, Q''\, \boldsymbol{N}$. Then, since $\mathcal{A}lg(P; Q') = \theta$, by Confluence Theorem and Lemma 3.11(i), there exists $\widetilde{Q}$, such that $Q'' \mapsto\!\!\!\!\rightarrow_\beta \widetilde{Q}$, and there exists θ', such that $\theta' = \mathcal{A}lg(P; \widetilde{Q})$, and $\theta \mapsto_\beta \theta'$. Thus, since by Lemma 3.12, $M\theta \mapsto\!\!\!\!\rightarrow_\beta M\theta'$, and $M\theta \in \mathsf{SN}^{\mathcal{O}}$, then also $M\theta' \in \mathsf{SN}^{\mathcal{O}}$. Hence, by induction hypothesis, $(\lambda P{:}\Delta.M)\, Q''\, \boldsymbol{N} \in \mathsf{SN}^{\mathcal{O}}$;
- $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N} \mapsto_\beta (\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N'}$, or $(\lambda P{:}\Delta.M)\, Q\, \boldsymbol{N} \mapsto_\beta (\lambda P{:}\Delta'.M)\, Q\, \boldsymbol{N}$. Then, the thesis follows by induction hypothesis.

Using a similar (simpler) argument, one can prove that $\mathsf{SN}^{\mathcal{O}}$ satisfies also (**c4**). □

Proof of Lemma 3.25. We prove a stronger statement for item (1), (we omit the proof of item (2), since it is similar): for any family A, and for any substitution θ, we have $[\![\, A\theta \,]\!]^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{O}}$. We proceed by induction on A.
• $A \equiv a\,\boldsymbol{N}$. Then, the thesis follows by definition of $[\![\ ]\!]^{\mathcal{F}}$, using Lemma 3.23.
• $A \equiv \checkmark P{:}\Delta{:}B.A'$. Then, $[\![\, A\theta \,]\!]^{\mathcal{F}} =$

$$\left\{ M \;\middle|\; Q \in [\![\, B \,]\!]^{\mathcal{F}} \;\Rightarrow\; M\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{ [\![\, A'\theta\overline{\theta} \,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q' \wedge \overline{\theta} = \mathcal{A}lg(P; Q') \} & \text{otherwise} \end{cases} \right\}$$

We have to check that $[\![\, A\theta \,]\!]^{\mathcal{F}}$ satisfies conditions (**c1**−**c4**) in Definition 3.22.
• (**c1**,**c2**) follow from the fact that, by induction hypothesis, $[\![\, A'\theta\overline{\theta} \,]\!]^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{O}}$, for all $\theta, \overline{\theta}$.
• (**c3**) We have to prove that $(\lambda P'{:}\Delta'.M')\, Q'\, \boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$, whenever there exists Q'', such that $Q' \mapsto\!\!\!\!\rightarrow_\beta Q''$, and $\overline{\theta} = \mathcal{A}lg(P'; Q'')$, and $(M'\overline{\theta})\, \boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$, and $\mathsf{CoDom}(\Delta'), Q' \in \mathsf{SN}^{\mathcal{O}}$. By definition of $[\![\, A\theta \,]\!]^{\mathcal{F}}$, we have $(\lambda P'{:}\Delta'.M')\, Q'\boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$ if, for any $Q \in [\![\, B \,]\!]^{\mathcal{F}}$,

$$(\lambda P'{:}\Delta'.M')\, Q'\, \boldsymbol{N'}\, Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{ [\![\, A'\theta\overline{\theta} \,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q'' \wedge \overline{\theta} = \mathcal{A}lg(P'; Q'') \} & \text{otherwise} \end{cases}$$

Thus, let Q be such that $Q \in [\![\, B \,]\!]^{\mathcal{F}}$, two cases can arise

(i) $P \not\sqsubseteq Q$. Since $(M'\overline{\theta})\, \boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$, by definition of $[\![\, A\theta \,]\!]^{\mathcal{F}}$, we have that $(M'\overline{\theta})\, \boldsymbol{N'}Q \in \mathsf{SN}^{\mathcal{O}}$, and since $\mathsf{SN}^{\mathcal{O}} \in \mathsf{Comp}^{\mathcal{O}}$, $\mathsf{SN}^{\mathcal{O}}$ satisfies condition (**c4**), and hence $(\lambda P'{:}\Delta'.M')\, Q'\boldsymbol{N'}\, Q \in \mathsf{SN}^{\mathcal{O}}$.

(ii) There exists Q'', such that $Q \mapsto\!\!\!\!\rightarrow_\beta Q''$, and $\overline{\theta} = \mathcal{A}lg(P'; Q'')$. Then, since $(M'\overline{\theta})\, \boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$, by definition of $[\![\, A\theta \,]\!]^{\mathcal{F}}$, we have $(M'\overline{\theta})\, \boldsymbol{N'}\, Q \in \bigcup\{ [\![\, A'\theta\overline{\theta} \,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q'' \wedge \overline{\theta} = \mathcal{A}lg(P'; Q'') \}$. Since, by induction hypothesis, $[\![\, A\theta\overline{\theta} \,]\!]^{\mathcal{F}}$ satisfies (**c3**), we have $(\lambda P'{:}\Delta'.M')\, Q'\ \boldsymbol{N'}\, Q \in \bigcup\{ [\![\, A'\theta\overline{\theta} \,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_\beta Q'' \wedge \overline{\theta} = \mathcal{A}lg(P'; Q'') \}$.

• (**c4**) Let $\mathsf{CoDom}(\Delta'), M', Q', \boldsymbol{N'} \in \mathsf{SN}$, and $P' \not\sqsubseteq Q'$. We have to prove that $(\lambda P'{:}\Delta'.M')\, Q'\, \boldsymbol{N'} \in [\![\, A\theta \,]\!]^{\mathcal{F}}$. To prove this, by definition of $[\![\, A\theta \,]\!]^{\mathcal{F}}$, it is sufficient to show the following two facts: let $Q \in [\![\, B \,]\!]^{\mathcal{F}}$, then

(i) if $P \not\sqsubseteq Q$, then $(\lambda P'{:}\Delta'.M')\, Q'\, \boldsymbol{N'}\, Q \in \mathsf{SN}^{\mathcal{O}}$;

(ii) otherwise, $(\lambda P{:}\Delta'.M')\,Q'\,\boldsymbol{N}'Q \in \bigcup\{[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_{\beta} Q'' \wedge \overline{\theta} = \mathcal{A}lg(P'; Q'')\}$.

Fact (1) above follows by the fact that $\mathsf{SN}^{\mathcal{O}} \in \mathsf{Comp}$ satisfies (**c4**). Fact (2) follows since, by induction hypothesis, each $[\![\, A\theta\overline{\theta}\,]\!]^{\mathcal{F}}$ also satisfies (**c4**).

Finally, let $A \equiv (\lambda P{:}\Delta.A')\,M\,\boldsymbol{N}$. Then,

$$[\![\, A\theta\,]\!]^{\mathcal{F}} = \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq M\theta \\ \bigcup\{[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}} \mid M\theta \mapsto\!\!\!\!\rightarrow_{\beta} M'' \wedge \overline{\theta} = \mathcal{A}lg(P'; M'')\} & \text{otherwise} \end{cases}$$

Now, one can easily check that $[\![\, A\theta\,]\!]^{\mathcal{F}}$ satisfies (**c1**−**c4**), by applying the induction hypothesis to $[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}}$. □

Proof of Lemma 3.26. We prove a stronger statement for item (1) (we omit the proof of item (2), which is similar): if $A \mapsto\!\!\!\!\rightarrow_{\beta} B$, and $\theta \mapsto\!\!\!\!\rightarrow_{\beta} \theta'$, then $[\![\, A\theta\,]\!]^{\mathcal{F}} = [\![\, B\theta'\,]\!]^{\mathcal{F}}$. We proceed by induction on the number of reduction steps of $A \mapsto\!\!\!\!\rightarrow_{\beta} B$.

Base case.

• $A \equiv B$. Then, we prove by induction on the structure of A that, if $\theta \mapsto\!\!\!\!\rightarrow_{\beta} \theta'$, then $[\![\, A\theta\,]\!]^{\mathcal{F}} = [\![\, A\theta'\,]\!]^{\mathcal{F}}$.

• $A \equiv a\boldsymbol{N}$. Then, the thesis is immediate.

• $A \equiv \checkmark P{:}\Delta{:}B.A'$. Then, $[\![\, (\checkmark P{:}\Delta{:}B.A')\theta\,]\!]^{\mathcal{F}} =$

$$\left\{ M \,\middle|\, Q \in [\![\, B\,]\!]^{\mathcal{F}} \;\Rightarrow\; M\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \wedge \overline{\theta} = \mathcal{A}lg(P; Q')\} & \text{otherwise} \end{cases} \right\}$$

and $[\![\, (\checkmark P{:}\Delta{:}B.A')\theta'\,]\!]^{\mathcal{F}} =$

$$\left\{ M \,\middle|\, Q \in [\![\, B\,]\!]^{\mathcal{F}} \;\Rightarrow\; M\,Q \in \begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq Q \\ \bigcup\{[\![\, A'\theta'\overline{\theta}\,]\!]^{\mathcal{F}} \mid Q \mapsto\!\!\!\!\rightarrow_{\beta} Q' \wedge \overline{\theta} = \mathcal{A}lg(P; Q')\} & \text{otherwise} \end{cases} \right\}$$

Now, from $\theta \mapsto\!\!\!\!\rightarrow_{\beta} \theta'$, using Lemma 3.12, we have $\theta\overline{\theta} \mapsto\!\!\!\!\rightarrow_{\beta} \theta'\overline{\theta}$. Thus, by induction hypothesis,

$$[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}} = [\![\, A'\theta'\overline{\theta}\,]\!]^{\mathcal{F}}$$

and hence the thesis follows immediately.

• $A \equiv (\lambda P{:}\Delta.A')M\boldsymbol{N}$, then

$[\![\, (\lambda P{:}\Delta\theta.A'\theta)(M\theta)(\boldsymbol{N}\theta)\,]\!]^{\mathcal{F}} =$

$$\begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq M\theta \\ \bigcup\{[\![\, (A'\theta\overline{\theta})(\boldsymbol{N}\theta)\,]\!]^{\mathcal{F}} \mid M\theta \mapsto\!\!\!\!\rightarrow_{\beta} M'' \wedge \overline{\theta} = \mathcal{A}lg(P; M'')\} & \text{otherwise} \end{cases}$$

and

$[\![\, (\lambda P{:}\Delta\theta.A'\theta')(M\theta')(\boldsymbol{N}\theta')\,]\!]^{\mathcal{F}} =$

$$\begin{cases} \mathsf{SN}^{\mathcal{O}} & \text{if } P \not\sqsubseteq M\theta' \\ \bigcup\{[\![\, (A'\theta'\overline{\theta}')(\boldsymbol{N}\theta')\,]\!]^{\mathcal{F}} \mid M\theta' \mapsto\!\!\!\!\rightarrow_{\beta} M'' \wedge \overline{\theta}' = \mathcal{A}lg(P; M'')\} & \text{otherwise} \end{cases}$$

Now, in order to show that $[\![\, A\theta\,]\!]^{\mathcal{F}} = [\![\, A\theta'\,]\!]^{\mathcal{F}}$, it is sufficient to prove the following fact:

Fact (*): whenever $M\theta \mapsto\!\!\!\!\rightarrow_{\beta} M''$ and $\overline{\theta} = \mathcal{A}lg(P; M'')$, then there exist M''' and $\overline{\theta}'$ such that $M\theta' \mapsto\!\!\!\!\rightarrow_{\beta} M'''$, $\overline{\theta}' = \mathcal{A}lg(P; M''')$, and $\overline{\theta} \mapsto\!\!\!\!\rightarrow_{\beta} \overline{\theta}'$.

Namely, if Fact (*) holds, then, by the Substitution Lemma 3.12, we have $\theta\overline{\theta} \mapsto\!\!\!\!\rightarrow_{\beta} \theta'\overline{\theta}'$, and, by induction hypothesis, $[\![\, A'\theta\overline{\theta}\,]\!]^{\mathcal{F}} = [\![\, A'\theta'\overline{\theta}'\,]\!]^{\mathcal{F}}$. But Fact (*) above follows from the Confluence Theorem 3.13 and Lemma 3.11.(i), using the fact that, by the Substitution Lemma 3.12, $M\theta \mapsto\!\!\!\!\rightarrow_{\beta} M\theta'$.

Induction Step.

• $A \mapsto_{\beta} B \mapsto\!\!\!\!\rightarrow_{\beta} B'$. Then, by induction hypothesis, $[\![\, B\theta'\,]\!]^{\mathcal{F}} = [\![\, B'\theta'\,]\!]^{\mathcal{F}}$. Thus, we are left to show that, if $A \mapsto_{\beta} B$, and $\theta \mapsto\!\!\!\!\rightarrow_{\beta} \theta'$, then $[\![\, A\theta\,]\!]^{\mathcal{F}} = [\![\, B\theta'\,]\!]^{\mathcal{F}}$. This is shown by induction on the structure of A.

• $A \equiv a\boldsymbol{N}$. Then, the thesis is immediate from the definition of $[\![\;\;]\!]^{\mathcal{F}}$.

• $A \equiv \checkmark P{:}\Delta.A' \mapsto_{\beta} \checkmark P{:}\Delta'.B'$. Then, the thesis follows by induction hypothesis, using an argument similar to that used for dealing with A of the same shape in the Base Case.

• $A \equiv (\lambda P{:}\Delta.A')M\boldsymbol{N}$. Then, there are two subcases:

(i) $A \equiv (\lambda P{:}\Delta.A')\,M\,\boldsymbol{N} \mapsto_{\beta} (\lambda P{:}\Delta.B')\,M'\,\boldsymbol{N}' \equiv B$;

(ii) $A \equiv (\lambda P{:}\Delta.A')\,M\,\boldsymbol{N} \mapsto_{\beta} A'\widehat{\theta}\boldsymbol{N} \equiv (A'\,\boldsymbol{N})\widehat{\theta} \equiv B$, where $\widehat{\theta} = \mathcal{A}lg(P; M)$.

In case (i), one can reason as in the Base Case. Let us prove the thesis in case (ii). We have:

$[\![\, A\theta\,]\!]^{\mathcal{F}} = \bigcup\{[\![\, (A'\boldsymbol{N})\theta\overline{\theta}\,]\!]^{\mathcal{F}} \mid M\theta \mapsto\!\!\!\!\rightarrow_{\beta} M'' \wedge \overline{\theta} = \mathcal{A}lg(P; M'')\}$ (*)

and

$[\![B\theta']\!]^{\mathcal{F}} = [\![(A'\ \boldsymbol{N})\widehat{\theta}\theta']\!]^{\mathcal{F}}$.

Using the Confluence Theorem 3.13, Lemma 3.11(i), and the induction hypothesis, one can show that all the elements in equation (*) above coincide. Moreover, by the Base Case, we have $[\![B\theta']\!]^{\mathcal{F}} = [\![B\theta]\!]^{\mathcal{F}} = [\![(A'\ \boldsymbol{N})\widehat{\theta}\theta]\!]^{\mathcal{F}}$. But then, since $\widehat{\theta} = \mathcal{A}lg(P; M)$, by Lemma 3.11(ii), we have that there exists $\overline{\theta} = \mathcal{A}lg(P; M\theta)$, such that $(A'\ \boldsymbol{N})\widehat{\theta}\theta \equiv (A'\boldsymbol{N})\theta\overline{\theta}$. Thus, $[\![B\theta']\!]^{\mathcal{F}} = [\![(A'\ \boldsymbol{N})\widehat{\theta}\theta]\!]^{\mathcal{F}} = [\![(A'\ \boldsymbol{N})\theta\overline{\theta}]\!]^{\mathcal{F}} = [\![A\theta]\!]^{\mathcal{F}}$. □

Proof of Lemma 3.27. We prove items (1), (2), (3) by mutual induction on the derivations of the judgments. We only deal with object rules, since the other rules can be dealt with similarly.

(O·Var) Immediate, since if $\Gamma_1, x{:}A, \Gamma_2 \vdash_\Sigma x : A$, then $x \notin \mathsf{Fv}(A)$.

(O·Const) Immediate.

(O·Conv) The thesis follows by induction hypothesis and by Lemma 3.26.

(O·Abs)

$$\frac{\Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma, \Delta \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma \lambda P{:}\Delta{:}B.A.M : \Pi P{:}\Delta.A}\ (O{\cdot}Abs)$$

We have to prove that $(\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].M[\boldsymbol{N}/\boldsymbol{x}]) \in [\![\Pi P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$ (1)

Let $Q \in [\![B]\!]$. Then statement (1) is true if the following two predicate are true:

(i) $\exists Q'.[Q \twoheadrightarrow_\beta Q' \wedge \mathcal{A}lg(P; Q') = \theta] \Longrightarrow (\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].M[\boldsymbol{N}/\boldsymbol{x}])\, Q \in [\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}}$;
(ii) $P \not\sqsubseteq Q \implies (\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].M[\boldsymbol{N}/\boldsymbol{x}])\, Q \in \mathsf{SN}^{\mathcal{O}}$.

• *Proof of (i).* By Lemma 3.25, $[\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}} \in \mathsf{Comp}^{\mathcal{O}}$, hence $[\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}}$ satisfies condition (**c3**) of Definition 3.22. Thus, for proving $(\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].M[\boldsymbol{N}/\boldsymbol{x}])\, Q \in [\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}}$, it is sufficient to prove that $\mathsf{CoDom}(\Delta[\boldsymbol{N}/\boldsymbol{x}])$, and $Q \in \mathsf{SN}$, and $M[\boldsymbol{N}/\boldsymbol{x}]\theta \in [\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}}$. Now, since $Q \in [\![B]\!]^{\mathcal{F}}$, then, by Lemma 3.25, we get $Q \in \mathsf{SN}^{\mathcal{O}}$. Moreover, $\mathsf{CoDom}(\Delta[\boldsymbol{N}/\boldsymbol{x}]) \in \mathsf{SN}^{\mathcal{F}}$, since by the Subderivation Property 3.14, for each family $A' \in \mathsf{CoDom}(\Delta)$, there exists a smaller derivation of $\Gamma' \vdash_\Sigma A' : K$; hence, we can apply the induction hypothesis to this latter derivation. Finally, $M[\boldsymbol{N}/\boldsymbol{x}]\theta \in [\![A[\boldsymbol{N}/\boldsymbol{x}]\theta]\!]^{\mathcal{F}}$, by induction hypothesis, noticing that $\mathsf{Dom}(\theta) = \mathsf{Dom}(\Delta)$.
• *Proof of (ii).* By induction hypothesis, $M[\boldsymbol{N}/\boldsymbol{x}] \in [\![A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$. Moreover, by Lemma 3.25, we get $[\![A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}} \subseteq \mathsf{SN}^{\mathcal{O}}$, hence in particular $M[\boldsymbol{N}/\boldsymbol{x}] \in \mathsf{SN}^{\mathcal{O}}$. Thus, since $\mathsf{SN}^{\mathcal{O}}$ is closed under (**c4**), using the Subderivation Property 3.14, we get $(\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].M[\boldsymbol{N}/\boldsymbol{x}])\, Q \in \mathsf{SN}^{\mathcal{O}}$.

(O·Appl)

$$\frac{\Gamma \vdash_\Sigma M_1 : \Pi P{:}\Delta.A \quad \Gamma, \Delta \vdash_\Sigma P : B \quad \Gamma \vdash_\Sigma M_2 : B}{\Gamma \vdash_\Sigma M_1\, M_2 : (\lambda P{:}\Delta.A)\, M_2}\ (O{\cdot}Appl)$$

We have to prove that $(M_1\, M_2)[\boldsymbol{N}/\boldsymbol{x}] \in [\![(\lambda P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].A[\boldsymbol{N}/\boldsymbol{x}])\, M_2[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$ (2) By induction hypothesis, we have $M_1[\boldsymbol{N}/\boldsymbol{x}] \in [\![\Pi P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$, with $P \in [\![B[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$, and $M_2[\boldsymbol{N}/\boldsymbol{x}] \in [\![B[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$. Now statement (2) follows by definition of $[\![\Pi P{:}\Delta[\boldsymbol{N}/\boldsymbol{x}].A[\boldsymbol{N}/\boldsymbol{x}]]\!]^{\mathcal{F}}$. □