The Rho Cube

Horatiu Cirstea, Claude Kirchner, and Luigi Liquori

LORIA INRIA INPL ENSMN
54506 Vandoeuvre-lès-Nancy BP 239 Cedex France
{Horatiu.Cirstea,Claude.Kirchner,Luigi.Liquori}@loria.fr
www.loria.fr/{~cirstea,~ckirchne,~lliquori}

Abstract. The *rewriting calculus*, or Rho Calculus (ρCal), is a simple calculus that uniformly integrates abstraction on patterns and non-determinism. Therefore, it fully integrates rewriting and lambda-calculus. The original presentation of the calculus was *untyped*. In this paper we present a uniform way to decorate the terms of the calculus with types. This gives raise to a new presentation *à la Church*, together with nine (8+1) type systems which can be placed in a ρ*-cube* that extends the λ-cube of Barendregt. Due to the matching capabilities of the calculus, the type systems use only one abstraction mechanism and therefore gives an original answer to the identification of the standard "λ" and "Π" abstractors.
As a consequence, this brings matching and rewriting as the first class concepts of the Rho-versions of the *Logical Framework* (LF) of Harper-Honsell-Plotkin, and of the *Calculus of Constructions* (CC) of Coquand-Huet.

1 Introduction

The first version of the simply typed lambda calculus was proposed by Church [Chu41] in order to build a calculus having a solid mathematical foundation and a clear relation with logics. Since then, many other type systems have been proposed, and a synthesis, known as the Barendregt's λ-cube, has been realized in [Bar92].

In the higher type theories, the arrow-types of the form $A \to B$ are replaced by dependent products of the form $\Pi X{:}A.B$ with B possibly containing X as a free variable. We have thus an abstraction not only at the level of terms (the "λ" operator) but an abstraction at the level of types as well (the "Π" operator).

Many works can be found in the literature [dB80,KN96,PM97,KBN99,KL01] on the subject of unifying the "λ" and "Π" operators. In particular, a goal of these researches is to replace the $\lambda{-}Appl$ rule below,whose meta-level application of substitution can be considered as non aesthetic, with the rule $\lambda{-}Appl'$:

$$\frac{\Gamma \vdash A : \Pi X{:}C.D \quad \Gamma \vdash B : C}{\Gamma \vdash AB : D[X/B]}\,(Appl{-}\lambda) \qquad \frac{\Gamma \vdash A : \Pi X{:}C.D \quad \Gamma \vdash B : C}{\Gamma \vdash AB : (\Pi X{:}C.D)B}\,(Appl'{-}\lambda)$$

and to use the reduction on Π-redexes, *i.e.*: $(\Pi X{:}C.D)B \to_{\beta\Pi} D[X/B]$ describing thus the Π-abstractor at the same level as the λ-abstractor. The four problems related to Π-reduction are clearly emphasized by Kamareddine *et alt.* [KN96,KBN99]:

1. Correctness of type is no longer valid;
2. The system is no longer safe;
3. The type of an expression may be not well-formed;
4. The Π-redexes are not well-formed.

In [KBN99], the authors proposed some solutions for use Π-reductions:

- *Infinite levels of abstraction*, *i.e.* using infinite levels of abstractors of the form "Λ^i", such that we are able to type a function $\Lambda^i X{:}A.B$ with a type $\Lambda^{i+1} X{:}A.C$, where C is the type of B. A similar approach is essentially inspired by the AUTOMATH project [dB80];
- *Abbreviations*, *i.e.* extending, in typing judgments, the standard contexts Γ with abbreviations of the form $(X{:}A)C$ and adding the rule $(Appl{-}\lambda\Pi)$: Intuitively, this rule says that if $B : D$ can be typed using the "abbreviation" that X of type A is C (*i.e.* $(X{:}A)C$), then $(\pi X{:}A.B)C : D[X/C]$ can be typed without this abbreviation.

$$\frac{\pi \in \{\lambda, \Pi\} \qquad \Gamma, (X{:}A)C \vdash B : D}{\Gamma \vdash (\pi X{:}A.B)C : D[X/C]}$$

In this paper we propose a different solution to the above four cited problems; our approach is essentially based on the *complete* unification of the two operators "λ" and "Π" into the only abstraction symbol present in the Rho Calculus, *i.e.* the "$\to$" operator. This unification is, so to speak, *built-in* in the definition of the Rho Calculus itself.

The rewriting calculus, or Rho Calculus, integrates in a uniform way matching, rewriting and non determinism. Its abstraction mechanism is based on the rewrite rule formation: in a ρ-term of the form $l \to r$, we abstract on the ρ-term l, and it is worth noticing that when l is a variable x this corresponds exactly to the λ-term $\lambda x.r$. When an abstraction $l \to r$ is applied to the ρ-term u, which is denoted $(l \to r)\bullet u$, the evaluation mechanism is based on the binding of the free variables present in l to the appropriate subterms of u. Indeed this binding is realized by matching l against u, and one of the characteristic of the calculus is to possibly use informations in the matching process such as algebraic axioms like associativity or commutativity. At that stage, non-determinism may come into play since the matching process can return zero, one, or several possibilities. For each of these variable bindings, the value of the variables are propagated in the term r yielding zero, one of several (finite or not) results. In a restricted way this is exactly what happen in the β-redex $(\lambda x.r)\, u$, which is simply denoted in the syntax of the Rho Calculus $(x \to r)\bullet u$, and where the match is trivially the substitution $\{x/u\}$. Therefore the Rho Calculus strictly contains the λ-calculus as the possibility to express failure of evaluation or multiplicity of results is directly supported.

The expressiveness of the arrow abstractor together with its matching powered evaluation rule makes very attractive the possibility to uniformly express both abstraction at the level of terms and of types. Therefore, we present and experiment a collection of nine $(8 + 1)$ type systems for the Rho Calculus decorated à la Church, that can be represented in a ρ-cube that extends the λ-cube of Barendregt. The most powerful type system in our cube (namely the ninth one) is a variant of the plain Calculus of Constructions and it is essentially inspired from the *Extended Calculus of Constructions* (ECC) of Z. Luo [Luo90]. In ECC, indeed, we have an infinite set of sorts, *i.e.* $s \in \{*, \Box_i\}$, with $i \in \mathbb{N}$, and the extra axiom $\vdash \Box_i : \Box_{i+1}$ with $i \in \mathbb{N}$. To sum up, we come up with a rich family of type systems which:

- provide typing mechanisms for the rewriting calculus;
- simplify and embeds the presentation of the λ-cube;
- provide an original and natural solution to the unification of the "λ" and "Π" operators;
- make matching power a truly integrated tool for type theory and therefore naturally integrates rewriting into type theory, and specifically into the calculus of construction;
- open new challenging questions and conjectures about a logics for the ρ-cube.

Road map of the paper: In Section 2, we present the syntax of the typed Rho Calculus, the matching relation, some simple matching theories, the operational semantics of the calculus, the type system, and a pictorial classification of the type systems in a ρ-cube. In Section 3, we present several examples of type derivations in some of the most interesting type systems. Section 4 presents some conjectures and we conclude the paper with some future works and perspectives.

The design and study of the Rho cube are ongoing works. In particular, we emphasize in the last part several questions and conjectures that we are currently working on. The reader will find on the web pages of the authors, the latest state of developments concerning these questions.

2 Syntax Matching and Types

We present here the syntax and operational semantics of the Rho Calculus. This extends with a very general type discipline the calculus presented in [CK99b,CKL00,Cir00].

2.1 Syntax

Notational Conventions. In this paper, the symbols A, B, C range over the set $\mathcal{T}$ of typed terms, the symbols $X^{\mathcal{T}}, Y^{\mathcal{T}}, Z^{\mathcal{T}}, \ldots$ range over the infinite set $\mathcal{V}$ of *typed* variables, the symbols $a^{\mathcal{T}}, b^{\mathcal{T}}, f^{\mathcal{T}} \ldots$ range over the infinite set $\mathcal{C}$ of *typed* constants, the symbol s ranges over the infinite set $\mathcal{S} \equiv \{*, \Box_i\}_{i \in N}$ of untyped sorts, and the symbols $\alpha^{\mathcal{T}}, \beta^{\mathcal{T}}, \ldots$ range over $\mathcal{C}$ and $\mathcal{V}$. All symbols can be indexed. The symbol $\equiv$ denotes syntactic identity of objects like terms or substitutions. We assume that variables and constants have a unique type, i.e. if $X^A, X^B \in \mathcal{V}$ (resp. $a^A, a^B \in \mathcal{C}$), then $A \equiv B$. An application of a constant function, say f^A, to a term B will be denoted by $f(B)$. We work modulo α-conversion, and we follow the Barendregt convention [Bar84], saying that free and bound variables have different names. The syntax of the pseudo-terms of the Rho Calculus are defined as follows:

$$\begin{array}{llr} \mathcal{T} ::= & \mathcal{S} \mid a^{\mathcal{T}} \mid X^{\mathcal{T}} \mid \mathcal{T} \to \mathcal{T} \mid \mathcal{T} \bullet \mathcal{T} \mid & \text{plain terms} \\ & \mathit{null} \mid \mathcal{T}, \mathcal{T} & \text{structured terms} \end{array}$$

The main intuition behind this syntax is that a rewrite rule $\mathcal{T} \to \mathcal{T}$ is an *abstraction*, the left-hand side of which determines the bound variables and some pattern structure. The application of a ρCal-term on another ρCal-term is represented by "$\bullet$". The terms can be grouped together into structures built using the "," operator and, according to the theory behind this operator, different structures can be obtained. We will come back to this later but, to support the intuition, let us mention that if we define the "," operator to be associative, then a *list structure* is obtained, while if the "," operator is specified as associative, commutative, and idempotent, then the *set structure* is obtained. The term *null* denotes an empty structure.

As usual, we assume that the application operator "$\bullet$" associates to the left while the "$\rightarrow$" and the "," operators associate to the right. The priority of the application "$\bullet$" is higher than that of the "$\rightarrow$" operator which is in turn of higher priority than the "," operator.

One may notice that in the syntax of ρCal, variables (free and bound) and constants are always annotated with a single and unique type with the only exception of the undecorated term *null*: as it will become clear when presenting the type system, this term can be assigned any type.

Example 1 (ρCal-terms). Some simple examples of ρCal-terms are:

- the term $(f^{A\rightarrow A}(X^A) \rightarrow X^A)\bullet f^{A\rightarrow A}(a^A)$ represents the application of the classical rewrite rule $f^{A\rightarrow A}(X^A) \rightarrow X^A$ to the term $f^{A\rightarrow A}(a^A)$;
- the term $X^A \rightarrow Y^B \rightarrow X^A$ is the typed lambda-term $\lambda X{:}A.\lambda Y{:}B.X$;
- the term (a^A, b^B, c^C) denotes a simple ternary structure;
- the term $(a_1^A \rightarrow b_1^B, a_2^A \rightarrow b_2^B)$ denotes a record with two fields, a_1 and a_2.

An important parameter of the ρCal is the matching theory $\mathbb{T}$. We assume given a theory $\mathbb{T}$, defined for example equationally, and we define matching problems in this general setting. In what follows we introduce several such theories, some of them being used later for instantiating the general ρCal.

2.2 Typed Matching Theories

An important parameter of the ρCal is the matching theory $\mathbb{T}$. We assume given a theory $\mathbb{T}$ and we define matching problems in this general setting. The theories extend the ones presented in [CKL00] in order to be compatible with the typing discipline. Matching theories are defined equationally via the judgment:

$$\Vdash A =_{\mathbb{T}} B : C$$

As we will see below, this judgment is defined in terms of the typing judgment $\vdash A : B$ defined in Subsection 2.4. Intuitively, its meaning is that A and B are terms of type C and are equivalent in the theory $\mathbb{T}$.

Definition 1 (Matching Theories).

- *the Empty theory $\mathbb{T}_\emptyset$ of equality is defined by the following inference rules:*

$$\frac{\Vdash A =_{\mathbb{T}_\emptyset} B : D \quad \Vdash B =_{\mathbb{T}_\emptyset} C : D}{\Vdash A =_{\mathbb{T}_\emptyset} C : D}\ (Trans) \qquad \frac{\Vdash A =_{\mathbb{T}_\emptyset} B : C}{\Vdash B =_{\mathbb{T}_\emptyset} A : C}\ (Symm)$$

$$\frac{\Vdash A =_{\mathbb{T}_\emptyset} B : E \quad \vdash C_{|p} : E \quad \vdash C : D}{\Vdash C[A]_p =_{\mathbb{T}_\emptyset} C[B]_p : D}\ (Ctx) \qquad \frac{\vdash A : B}{\Vdash A =_{\mathbb{T}_\emptyset} A : B}\ (Refl)$$

where $A[B]_p$ denotes the term A with the term B at position p, and $A_{|p}$ the term at the position p of the term A.

Note that in the theory $\mathbb{T}_\emptyset$ we do not need an inference rule describing the stability by substitution since all the below theories are defined using inference rules exhibiting all possible instantiations for the terms.

– *for a given binary symbol f, the theory of Commutativity $\mathbb{T}_{C(f)}$ is defined by $\mathbb{T}_\emptyset$ plus the following inference rule:*

$$\frac{\vdash f^{C\to C\to D}(A\ B) : D}{\Vdash f^{C\to C\to D}(A\ B) =_{\mathbb{T}_{C(f)}} f^{C\to C\to D}(B\ A) : D}\ (Comm)$$

– *for a given binary symbol f, the theory of Associativity $\mathbb{T}_{A(f)}$ is defined by $\mathbb{T}_\emptyset$ plus the following inference rule:*

$$\frac{\vdash f^{D\to D\to D}(f^{D\to D\to D}(A\ B)\ C) : D}{\Vdash f^{D\to D\to D}(f^{D\to D\to D}(A\ B)\ C) =_{\mathbb{T}_{A(f)}} f^{D\to D\to D}(A\ f^{D\to D\to D}(B\ C)) : D}\ (Assoc)$$

– *for a given binary symbol f, the theory of Idempotency $\mathbb{T}_{I(f)}$ is defined by $\mathbb{T}_\emptyset$ plus the following inference rule:*

$$\frac{\vdash f^{B\to B\to B}(A\ A) : B}{\Vdash f^{B\to B\to B}(A\ A) =_{\mathbb{T}_{I(f)}} A : B}\ (Idem)$$

– *for a given binary symbol f and a constant 0, the theory of Neutral (Right and Left) element $\mathbb{T}_{N(f^0)}$ is defined by $\mathbb{T}_\emptyset$ plus the following inference rules:*

$$\frac{\vdash f^{B\to B\to B}(0^B\ A) : B}{\Vdash f^{B\to B\to B}(0^B\ A) =_{\mathbb{T}_{N(f^0)}} A : B}\ (0_L) \qquad \frac{\vdash f^{B\to B\to B}(A\ 0^B) : B}{\Vdash f^{B\to B\to B}(A\ 0^B) =_{\mathbb{T}_{N(f^0)}} A : B}\ (0_R)$$

– *the MultiSet theory, $\mathbb{T}_{MSet(f,nil)}$, is obtained by considering the symbol f as an associative and commutative function symbol with nil as neutral element:*

$$\mathbb{T}_{MSet(f,nil)} = \mathbb{T}_{A(f)} \cup \mathbb{T}_{C(f)} \cup \mathbb{T}_{N(f^{nil})}$$

– *the Set theory, $\mathbb{T}_{Set(f,nil)}$, is obtained from $\mathbb{T}_{MSet(f,nil)}$, by considering the symbol f as an idempotent function symbol:*

$$\mathbb{T}_{Set(f,nil)} = \mathbb{T}_{MSet(f,nil)} \cup \mathbb{T}_{I(f)}$$

As already mentioned, computing the substitutions which solve the matching (in the theory $\mathbb{T}$) from a pattern A to a subject B is a fundamental element of the ρCal. Syntactic matching is formally defined as follows:

Definition 2 (Syntactic Matching). *For a given theory $\mathbb{T}$ over ρCal-terms:*

1. *a* $\mathbb{T}$-match equation *is a formula of the form* $A \ll_{\mathbb{T}} B$*;*
2. *a* substitution σ *is a solution of the* $\mathbb{T}$*-match equation* $A \ll_{\mathbb{T}} B$ *if* $\sigma A =_{\mathbb{T}} B$*;*
3. *a* $\mathbb{T}$-matching system *is a conjunction of* $\mathbb{T}$*-match equations;*
4. *a substitution* σ *is a solution of a* $\mathbb{T}$*-matching system if it is a solution of all the* $\mathbb{T}$*-match equations in it;*
5. *a* $\mathbb{T}$*-matching system is* trivial *when all substitutions are solution of it and we denote by* $\mathbb{F}$ *a* $\mathbb{T}$*-matching system without solution;*
6. *we define the function* $\mathcal{S}ol$ *on a* $\mathbb{T}$*-matching system* $\mathcal{S}$ *as returning the* $\prec$*-ordered*[1] *list of all* $\mathbb{T}$*-matches of* $\mathcal{S}$ *when* $\mathcal{S}$ *is not trivial and the list containing only* $\sigma_{\mathbf{id}}$*, where* $\sigma_{\mathbf{id}}$ *is the identity substitution, when* $\mathcal{S}$ *is trivial.*

Notice that when the matching algorithm fails to find a solution (*i.e.* returns $\mathbb{F}$), the function $\mathcal{S}ol$ returns the empty list.

Since in general we can consider arbitrary theories over ρCal-terms, $\mathbb{T}$-matching is in general undecidable, even when restricted to first-order equational theories [JK91].

For example, in $\mathbb{T}_\emptyset$, the matching substitution from a ρCal-term A to a ρCal-term B can be computed by the rewrite system presented in Figure 1, where the symbol $\wedge$ is assumed to be associative and commutative, and $\diamond_1$, $\diamond_2$ are either constant symbols or the prefix notations of "," or "•" or "$\rightarrow$". Starting

$$
\begin{array}{lcll}
\diamond_1(A_1 \ldots A_n) \ll_{\mathbb{T}_\emptyset} \diamond_2(B_1 \ldots B_m) & \rightsquigarrow & \begin{cases} \bigwedge\limits_{i=1\ldots n} A_i \ll_{\mathbb{T}_\emptyset} B_i & \text{if } \diamond_1 \equiv \diamond_2 \text{ and } n = m \\ \mathbb{F} & \text{otherwise} \end{cases} \\
(X^C \ll_{\mathbb{T}_\emptyset} A) \wedge (X^C \ll_{\mathbb{T}_\emptyset} B) & \rightsquigarrow & \begin{cases} X^C \ll_{\mathbb{T}_\emptyset} A & \text{if } \Vdash A =_{\mathbb{T}_\emptyset} B : C \\ \mathbb{F} & \text{otherwise} \end{cases} \\
A \ll_{\mathbb{T}_\emptyset} X^B & \rightsquigarrow & \mathbb{F} & \text{if } A \notin \mathcal{V} \\
\mathbb{F} \wedge (A \ll_{\mathbb{T}_\emptyset} B) & \rightsquigarrow & \mathbb{F}
\end{array}
$$

Fig. 1. Rules for Typed Syntactic Matching

from a matching system $\mathcal{S}$, the application of this rule set terminates and returns either $\mathbb{F}$ when there are no substitutions solving the system, or a system $\mathcal{S}'$ in "normal form" from which the solution can be trivially inferred [KK99]. This set of rules could be easily extended to matching modulo commutativity (it is

[1] We consider a total order $\prec$ on the set of substitutions. Such orderings always exist (see [CKL00]): one can for example take the lexicographic ordering on the flattened representation of the substitutions. The ordering on substitutions can be seen as a parameter of the ρCal and thus can be customized by the user.

decidable too but the number of matches can then be exponential in the size of the initial problem). It is also decidable for associativity-commutativity [Hul79] but in the general case matching could be as difficult as unification [Bür89].

Example 2 (Syntactic Matching).

1. in $\mathbb{T}_\emptyset$, the equation $a^A \ll_{\mathbb{T}_\emptyset} b^B$ has no solution, and thus $\mathcal{S}ol(a^A \ll_{\mathbb{T}_\emptyset} b^B)$ returns the empty list of substitutions;
2. in $\mathbb{T}_\emptyset$, $a^A \ll_{\mathbb{T}_\emptyset} a^A$ is solved by all substitutions, and thus $\mathcal{S}ol(a^A \ll_{\mathbb{T}_\emptyset} a^A)$ returns the list with only one element, $\sigma_{\mathbf{id}}$;
3. in $\mathbb{T}_\emptyset$, the equation $f(X^A\ \ g(X^A\ \ Y^D)) \ll_{\mathbb{T}_\emptyset} f(a^A\ \ g(a^A\ \ d^D))$ (f, g stand for $f^{A\to B\to C}, g^{A\to D\to B}$) has as solution the substitution $\sigma \equiv [X^A/a^A\ \ Y^D/d^D]$;
4. in $\mathbb{T}_{C(f)}$ the equation $f(X^A\ Y^A) \ll_{\mathbb{T}_{C(f)}} f(a_1^A\ a_2^A)$ (f stands for the a commutative symbol $f^{A\to A\to B}$) has the two solutions $\sigma_1 \equiv [X^A/a_1^A\ \ Y^A/a_2^A]$ and $\sigma_2 \equiv [X^A/a_2^A\ \ Y^A/a_1^A]$, and therefore the matching equation $\mathcal{S}ol(f(X^A\ Y^A) \ll_{\mathbb{T}_{C(f)}} f(a_1^A\ a_2^A))$ is solved by the list $\sigma_1\ \sigma_2$;
5. in $\mathbb{T}_{AN} = \mathbb{T}_{A(,)} \cup \mathbb{T}_{N(,^{null})}$, the equation $(X^A, a^A, Y^A) \ll_{\mathbb{T}_{AN}} (a_1^A, a^A, a_2^A, a_3^A)$ has as solution the substitution $\sigma \equiv [X^A/a_1^A\ \ Y^A/(a_2^A, a_3^A)]$, while the two equations $(X^A, a^A, Y^A) \ll_{\mathbb{T}_{AN}} (null, a^A)$ and $(X^A, a^A, Y^A) \ll_{\mathbb{T}_{AN}} a^A$ have both the same solution $\sigma \equiv [X^A/null\ \ Y^A/null]$, since $\Vdash a^A =_{\mathbb{T}_{AN}} (null, a^A) =_{\mathbb{T}_{AN}} (null, a^A, null) : A$;
6. in $\mathbb{T}_{ACN} = \mathbb{T}_{A(,)} \cup \mathbb{T}_{C(,)} \cup \mathbb{T}_{N(,^{null})}$, the equation $(X^A, a^A) \ll_{\mathbb{T}_{ACN}} (a_1^A, a^A, a_2^A, a_2^A)$ has as solution the substitution $\sigma \equiv [X^A/(a_1^A, a_2^A, a_3^A)]$, since $\Vdash (a_1^A, a^A, a_2^A, a_3^A) =_{\mathbb{T}_{ACN}} (a_1^A, a_2^A, a_3^A, a^A) : A$.

2.3 Operational Semantics

Using the same notations as [CKL00] to which the reader is referred for further discussion of the concepts, the notion of reduction on terms in $\mathbb{T}$ is uniformly defined as follows: for a given ordering $\prec$ on substitutions (which is left implicit in the notation), and a theory $\mathbb{T}$, the operational semantics is defined by the computational rules given in Figure 2. The central idea of the main rule of the

$$\begin{array}{lll}
(\rho)\ (A \to B)\bullet C & \mapsto_{\mathbb{T}} & \begin{cases} null & \text{if } A \ll_{\mathbb{T}} C \text{ has no solution} \\ \sigma_1 B, \ldots, \sigma_n B & \text{if } \sigma_i \in Sol(A \ll_{\mathbb{T}} C),\ \sigma_1 \prec \sigma_{i+1},\ n \leq \infty \end{cases} \\
(\epsilon)\quad (A, B)\bullet C & \mapsto_{\mathbb{T}} & A\bullet C, B\bullet C \\
(\nu)\quad null\bullet A & \mapsto_{\mathbb{T}} & null
\end{array}$$

Fig. 2. Evaluation rules of the ρCal.

calculus (ρ) is that the application of a rewrite rule $A \to B$ at the root (also called top) position of a term C, consists in computing all the solutions of the matching equation ($A \ll_{\mathbb{T}} C$) in the theory $\mathbb{T}$ and applying all the substitutions from the $\prec$-ordered list returned by the function $Sol(A \ll_{\mathbb{T}} C)$ to the term

B. When there is no solution for the matching equation ($A \ll_{\mathbb{T}} C$) the special constant *null* is obtained as result of the application. Notice that there could be an infinity of solutions to the matching problem ($A \ll_{\mathbb{T}} C$), but in this paper we restrict ourselves to the case where the matching problem is finitary.

It is important to remark that if A is a variable, then the (ρ) rule corresponds exactly to the (β) rule of the lambda-calculus, and variables manipulations in substitutions take places externally, using α-conversion and Barendregt's convention if necessary.

The rules (ϵ) and (ν) deal with the distributivity of the application on the structures whose constructors are "," and *null*.

With respect to the previous presentation of the Rho Calculus [CK99b,CK99a], we have modified the notation of the application operator, simplified the evaluation rules, and generalised to deal with generic result structures.

When the theory $\mathbb{T}$ is clear from the context, its denotation will be omitted.

Definition 3. *We denote by $=_\rho$ the reflexive, symmetric, transitive, and contextual closure of the reduction $\mapsto_{\mathbb{T}}$ over a theory $\mathbb{T}$. The relation $=_\rho$ is a congruence relation.*

When working modulo reasonably powerful theories $\mathbb{T}$, the evaluation rules of the ρCal are confluent [Cir00,CKL00].

Theorem 1 (Confluence in $\mathbb{T}_\emptyset$). *Given a term A such that all its abstractions contain no arrow in the first argument, if $A \mapsto\!\!\!\twoheadrightarrow_{\mathbb{T}_\emptyset} B$ and $A \mapsto\!\!\!\twoheadrightarrow_{\mathbb{T}_\emptyset} C$ then there exists a term D such that $B \mapsto\!\!\!\twoheadrightarrow_{\mathbb{T}_\emptyset} D$ and $C \mapsto\!\!\!\twoheadrightarrow_{\mathbb{T}_\emptyset} D$.*

2.4 The Type System

A statement is of the form $A : B$, with $A, B \in \mathcal{T}$. Judgements are defined as $\vdash A : B$, *i.e.* without any kind of context information: this was possible thanks to the explicit type-decoration for *all* constants and variables (generically denoted by α). We call A the subject and B the predicate of the judgment, respectively. The type rules in Figure 3 are given in three groups: general rules for applications/structures (common to all systems), and one specific rule schema (characteristic to each system) parameterized by a suitable choice of sorts.

Our type system presents in fact nine type systems instead of the usual eight present in the λ-cube; the eight type systems of the ρ-cube (natural counterparts of the λ-cube) are defined by taking the general rules plus the specific subset of rules $\{(*,*), (*,\Box_0), (\Box_0,*), (\Box_0,\Box_0)\}$. In the more general case, many other subsets of specific rules can be obtained by considering all the following pairs of sorts: $\{(*,*), (*,\Box_i), (\Box_i,*), (\Box_i,\Box_j)\}$ with $i, j \in \mathbb{N}$. This (possibly infinite) subset of rules gives raise to the most powerful type system ρECC, which uses the full typing power of the ECC of Z. Luo. Figure 3 describes also all the systems and collects it in a graphical ρ-cube. The system $\rho \rightarrow$ is essentially a variant of the system presented in [Cir00,CK00].

In order to help the reader in understanding the type system, we give here a small explication for all the type rules and a brief comparison with the rules of the λ-cube:

General Rules for Applications

$$\frac{}{\vdash * : \Box_0}\ (Axiom_*) \qquad \frac{i \in \mathbb{N}}{\vdash \Box_i : \Box_{i+1}}\ (Axiom_\Box) \qquad \frac{\vdash A : s}{\vdash \alpha^A : A}\ (Start)$$

$$\frac{\vdash B : C \quad \vdash A \to C : s}{\vdash A \to B : A \to C}\ (Abs) \qquad \frac{\vdash A : C \to D \quad \vdash C : E \quad \vdash B : E}{\vdash A{\bullet}B : (C \to D){\bullet}B}\ (Appl)$$

$$\frac{\vdash A : C \quad \vdash B : s \quad B =_\rho C}{\vdash A : B}\ (Conv)$$

General Rules for Structures

$$\frac{\vdash A : s}{\vdash null : A}\ (Null) \qquad \frac{\vdash A : C \quad \vdash B : C}{\vdash A, B : C}\ (Struct)$$

Specific Rules

$$\frac{\vdash A : C \quad \vdash C : s_1 \quad \vdash B : s_2}{\vdash A \to B : s_2}\ (s_1, s_2)$$

System	Set of specific rules ($i, j \in \mathbb{N}$)
$\rho \to$	$(*,*)$
$\rho 2$	$(*,*)$ $(\Box_0,*)$
$\rho P = \rho LF$	$(*,*)$ $(*,\Box_0)$
$\rho P2$	$(*,*)$ $(\Box_0,*)$ $(*,\Box_0)$
$\rho\underline{\omega}$	$(*,*)$ $(\Box_0,\Box_0)$
$\rho\omega$	$(*,*)$ $(\Box_0,*)$ $(\Box_0,\Box_0)$
$\rho P\omega$	$(*,*)$ $(*,\Box_0)$ $(\Box_0,\Box_0)$
$\rho P\underline{\omega} = \rho CC$	$(*,*)$ $(\Box_0,*)$ $(*,\Box_0)$ $(\Box_0,\Box_0)$
ρECC	$(*,*)$ $(\Box_i,*)$ $(*,\Box_i)$ $(\Box_i,\Box_j)$

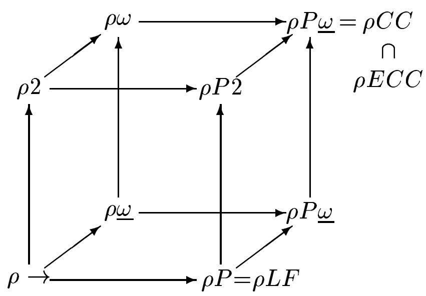

Fig. 3. The type system rules, the generic rules, and the ρ-cube

- $(Axiom_*), (Axiom_\Box)$. Those two rules are used in the leaves of any well-typed derivation; in particular the rule $(Axiom_\Box)$ allows one to obtain a *type hierarchy* similar to that of Martin-Löf type theory [Mar84];
- $(Start)$. Only well decorated constants of variables are well typed: note that the type is "written" in the term itself, as for example in X^A, or a^A;
- (Abs). This rule represents the counterpart of the λ-cube rule $(Abs{-}\lambda)$ which could be written in our syntax as $(Abs{-}\lambda\rho)$:

$$\frac{\Gamma, X{:}D \vdash B : C \quad \Gamma \vdash \Pi X{:}D.C : s}{\Gamma \vdash \lambda X{:}D.B : \Pi X{:}D.C}\ (Abs{-}\lambda) \qquad \frac{\vdash B : C \quad \vdash X^D \to C : s}{\vdash X^D \to B : X^D \to C}\ (Abs{-}\lambda\rho)$$

The main differences comparing to the corresponding rule in the ρ-cube are:

1. the λ and Π-operators are unified into the $\to$ operator: in fact the Π-operator can be considered as a limited form of the $\to$ abstractor, where the pattern used for the matching is always a variable X^D. In the (Abs) rule the pattern A can be more sophisticated than a simple variable and hence the result type of an abstraction is $A \to C$;
2. since all variables (free and bound) are decorated by their type, the notion of context Γ is useless, and therefore we do not need in the first premise of the rule to enrich the context Γ with the declaration of the bound variable X^D;
3. as in the λ-cube, the second premise is a call to a specific rule (s_1, s_2) that allows one to discern between the different type systems of the ρ-cube;

– $(Appl)$. This rule represents the counterpart of the λ-cube rule $(Appl{-}\lambda)$ which could be written in our syntax as $(Appl{-}\lambda\rho)$:

$$\frac{\Gamma \vdash A : \Pi X{:}E.D \quad \Gamma \vdash B : E}{\Gamma \vdash AB : D[X/B]}\ (Appl{-}\lambda) \qquad \frac{\vdash A : X^E \to D \quad \vdash B : E}{\vdash A \bullet B : D[X/B]}\ (Appl{-}\lambda\rho)$$

The main differences comparing to the corresponding rule in the ρ-cube are:
1. in the first premise, the term A is typed with a term $C \to D$ instead of a Π-term, where C can be *any* ρ-term;
2. in order to type-check the type of the argument with the domain-type of the function, we need to check both the type of the pattern C (since it is not necessarily a variable) and of the argument B; this task is done by the second and the third premises;
3. the result type of the application will be in turn the type $(C \to D)\bullet B$, *i.e.* a type not in normal-form. Note here the difference with respect to the solution adopted in the λ-cube, where the result type was $D[X/B]$: this solution takes into account, in a very natural way, all the type systems placed in the right-hand side of the ρ-cube, *i.e.* the ones with dependent-types, and enlightens the higher-order nature of the $\to$ operator;

– $(Conv)$. This rule (not syntax-directed) has the same shape as the correspondent rule in the λ-cube. When building derivations, this rule should be used in order to reduce the type $(C \to D)\bullet B$ to an equivalent type;
– $(Null)$. The untyped constant *null* is typed with any (well-formed) type;
– $(Struct)$. This rule for structures enforces the type of A to be the same as the type of B; essentially we guarantee that all the terms in a structure have exactly the same type. A more flexible rule that deals with terms having a different type in the same structure would be $(Struct')$.

This rule would be essential for typing the *object-oriented* features of the ρCal ([CKL00]). This topic is subject of a current parallel study;

$$\frac{\vdash A : C \quad \vdash B : D}{\vdash A, B : C, D}\ (Struct')$$

This rule would be essential for typing the *object-oriented* features of the ρCal presented in [CKL00]. This topic is subject of a current parallel study;

– (s_1, s_2). This rule is usually invoked in order to demonstrate the second premise of the (Abs) rule: it is a rule schema since it represents in fact an infinite set of type rules, depending on the shape of the sorts s_1 and s_2.

Remark. Note that the more general type system ρECC differs from the original Luo's ECC since we do not introduce:

1. Σ-types (or, strong *sum-types*) $\Sigma X{:}A.B$;
2. a *type-cumulativity* order $\preceq$ on types, re-paraphrased in the ρCal as the smallest partial order over ρ-terms w.r.t. conversion $\simeq$[2] such that:
   (a) $* \preceq \Box_0 \preceq \Box_1 \preceq \ldots$ and (b) $A \simeq A' \wedge B \preceq B' \Longrightarrow A \to B \preceq A' \to B'$;
3. the "subtyping-like" rule (cum) $(\vdash A : B \;\&\; \vdash C : \Box_i \;\&\; B \preceq C \;\Longrightarrow \vdash A : C)$.

3 Examples

In this section we present some judgments and (for lack of space) one type derivation in some representative type systems of the ρ-cube, namely:

- in the system $\rho\to$, the simply typed Rho Calculus (terms dependent on terms);
- in the system $\rho 2$, *i.e.* the polymorphic (in the sense of Girard's system F ($\lambda 2$) [Gir86]) Rho Calculus (terms dependent on types);
- in the system ρLF, *i.e.* the simply typed Rho Calculus with dependent-types: this system is the counterpart of the system LF, *i.e.* the *Logical Framework* [HHP92] (types dependent on terms);
- in the system $\rho\underline{\omega}$, (types depending on types);
- in the system ρCC, *i.e.* the counterpart of the system ECC of Z. Luo.

Most of the examples are inspired (or re-arranged) from [Bar92]. When not explicitly mentioned, we denote the symbol $\Box_0$ by $\Box$.

Example 3 (The Type System $\rho\to$). In this system the following can be derived when we consider the constants int^* and 3^{int^*}:

$$\dfrac{\dfrac{\vdash X^{int^*} : int^* \qquad \dfrac{\vdash X^{int^*} : int^* \quad \vdash int^* : *}{\vdash X^{int^*} \to int^* : *}}{\vdash X^{int^*} \to X^{int^*} : X^{int^*} \to int^*} \qquad \vdash X^{int^*} : int^* \quad \vdash 3^{int^*} : int^*}{\dfrac{\vdash (X^{int^*} \to X^{int^*})\bullet 3^{int^*} : (X^{int^*} \to int^*)\bullet 3^{int^*} \qquad (1,2)}{\vdash (X^{int^*} \to X^{int^*})\bullet 3^{int^*} : int^*}}$$

where (1) is $\vdash int^* : *$, and (2) is $(X^{int^*} \to int^*)\bullet 3^{int^*} =_\rho int^*$, and the last applied rule is $(Conv)$. Proceeding in a similar way, it is not difficult to derive in $\rho\to$ (we write f, g, h instead of $f^{A\to A}, g^{A\to A}, h^{A\to A\to A}$):

$$\begin{array}{ll} \vdash X^A \to A : * & \vdash (f(X^A) \to X^A, g(X^A) \to X^A)\bullet f(a^A) : A \\ \vdash (f(X^A) \to X^A)\bullet f(a_1^A) : A & \vdash (h(X^A\ Y^A) \to (X^A, Y^A))\bullet h(a_1^A\ a_2^A) : A \end{array}$$

[2] $\simeq$ denotes the identity obtained w.r.t. the reflexivity and antisymmetry of $\preceq$.

Example 4 (The Type System ρLF). In this system the following can be derived:

$$\dfrac{\dfrac{\dfrac{\vdash X^{int^*} : int^* \quad \vdash int^* : * \quad \vdash * : \Box}{\vdash X^{int^*} \to * : \Box}}{\vdash f^{X^{int^*} \to *} : X^{int^*} \to * \quad \vdash X^{int^*} : int^* \quad \vdash 3^{int^*} : int^*}}{\dfrac{\vdash f^{X^{int^*} \to *} \bullet 3^{int^*} : (X^{int^*} \to *) \bullet 3^{int^*} \qquad (1)\ \ (X^{int^*} \to *) \bullet 3^{int^*} =_\rho *}{\vdash f^{X^{int^*} \to *}(3^{int^*}) \equiv f^{X^{int^*} \to *} \bullet 3^{int^*} : *}}$$

where (1) is $\vdash * : \Box$ the last applied rule is $(Conv)$. Observe that the judgment $\vdash X^{int^*} \to * : \Box$ can be derived thanks to the specific rule $(*, \Box)$.

In a similar way, we can easily derive the following judgments (here p, q are used in $\vdash p^{A \to *}(X^A) : *$ and $\vdash q^{A \to *}(X^A) : *$[3])

$$\begin{array}{ll} \vdash X^A \to Y^{p(X^A)} \to q(X^A) : *^4 & \vdash X^A \to Y^{p(X^A)} \to p(X^A) : *^5 \\ \vdash X^A \to Y^{p(X^A)} \to Y^{p(X^A)} : X^A \to Y^{p(X^A)} \to p(X^A)^6 & \end{array}$$

Example 5 (The Type Systems $\rho 2$, $\rho\underline{\omega}$, ρCC).

$$\begin{array}{ll} \text{in } \rho 2 & \vdash \bot \equiv X^* \to X^* : * \\ & \vdash Y^* \to Z^\bot \to (Z^\bot \bullet Y^*) : Y^* \to Z^\bot \to Y^{*\,7} \\ & \vdash X^* \to Y^{X^*} \to Y^{X^*} : X^* \to Y^{X^*} \to X^* \text{ (the polymorphic identity)} \\ & \vdash (X^* \to f(Y^{X^*}) \to Y^{X^*}) \bullet int^* \bullet f(3^{int^*}) : int^* \vdash X^* \to * : \Box \\ \text{in } \rho\underline{\omega} & \vdash X^* \to X^* : X^* \to * \text{ and } \vdash (X^* \to X^*) \bullet int^* : (X^* \to *) \bullet int^* =_\rho * \\ \text{in } \rho CC & \vdash X^\Box \to \Box : \Box_1 \text{ and } \vdash X^\Box \to X^\Box : X^\Box \to \Box \\ & \vdash (X^\Box \to X^\Box) \bullet Y^\Box : (X^\Box \to \Box) \bullet Y^\Box =_\rho \Box \end{array}$$

3.1 The Failure of the Unicity of Typing

When dealing with typed calculi, one interesting property is the *unicity of typing* (up to conversion), which can be rephrased in our Rho Calculus as follows:

$$\vdash A : B \ \wedge \ \vdash A : B' \implies B =_\rho B'$$

This property is verified in all systems defining λ-cubes (or derivates); in ECC [Luo90], thanks to the type cumulability relation and the (cum) type rule, a

[3] Following the Curry-Howard isomorphism, if A is considered as a set, $X \in A$, and p, q are predicates on A, then $p(X^A), q(X^A)$ are types, considered as propositions.

[4] This proposition states that the predicate p (on A) considered as a set is included in the predicate q.

[5] This proposition states the reflexivity of the inclusion.

[6] The subject of this statement gives a "proof" of the reflexivity of the inclusion.

[7] In this case, the predicate of the judgment considered as proposition says: *ex falso sequitur quodlibet*, *i.e.* anything follows from a false judgment: the subject of this judgment is its proof.

weaker property, *i.e.* the *minimum type property*, holds:

$$\exists B!\, \forall B' \quad \vdash A : B \ \wedge \ \vdash A : B' \implies B \preceq B'$$

The type systems in the right-hand side of the ρ-cube does not satisfies those two properties. If we take, for example in ρLF, the ρ-term $X^{Y^*} \to Y^*$, can be typed with both $*$ and $X^{Y^*} \to *$. Since in the Rho Calculus the $\to$ unifies the λ-cube symbols "λ" and "Π", it is natural to interpret the term $\vdash X^{Y^*} \to Y^*$ in two different ways:

$$\vdash X^{Y^*} \to Y^* : \begin{cases} * & \textit{i.e.} \text{ the term is an "arrow-type"} \\ X^{Y^*} \to * & \textit{i.e.} \text{ the term is a "constructor for dependent-types"} \end{cases}$$

In fact the translation of the above term into the λ-cube is ambiguous:

$$[\![X^{Y^*} \to Y^*]\!] = \begin{cases} \Pi X{:}Y.Y \text{ and } Y : * \vdash_{\lambda\to} \Pi X{:}Y.Y : * \\ \lambda X{:}Y.Y \text{ and } Y : * \vdash_{\lambda LF} \lambda X{:}Y.Y : \Pi X{:}Y.* \end{cases}$$

4 Some Conjectures and Conclusion

4.1 Conjectures

The full proof-theoretical work for the ρ-cube has to be done: however, to stimulate the interested reader, we would like to sketch some conjectures that should be proved in a future work. We denote $\vdash_{\mathcal{S}}$ as the symbol of derivability in one of the type systems $\mathcal{S}$ of the ρ-cube.

Property 1 (Consistency). The ρ-cube is logically consistent, *i.e.* given a closed term A, we have $\not\vdash_{\mathcal{S}} A : \bot$.

Property 2 (Correctness of Typing (in ρECC)). If $\vdash_{\mathcal{S}} A : B$, then $\vdash_{\rho ECC} B : C$.

This property states in fact that we can find a correct derivation in ρECC for any predicate of a judgment in $\mathcal{S}$. It is crucial to prove subject reduction.

Property 3 (Weak Uniqueness of Typing). If $\vdash_{\mathcal{S}} A : B$ and $\vdash_{\mathcal{S}} A : B'$ and $B \neq_\rho s \neq_\rho B'$, then $B =_\rho B'$.

This property states that, for the simple terms of the ρCal, we can find a principal type. It is crucial for the decidability of type checking.

Property 4 (Subject Reduction (in ρECC)). If $\vdash_{\mathcal{S}} A : B$, and $A \mapsto\!\!\!\!\twoheadrightarrow C$, then $\vdash_{\rho ECC} C : B$.

This property proves that types are preserved over reduction.

Property 5 (Strong Normalization of Typable Terms). If $\vdash_{\mathcal{S}} A : B$, then A and B are strongly normalizing.

This property states that every typable term normalizes. It will be proved following either the lines of [CH88,Coq91], or in a modular fashion, by first translating the result of strong normalization for $\lambda\omega$ [Gir72] into $\rho\omega$, and then using a translation function from $\rho\omega$ into ρECC, in the style of [HHP92,GN91].

Property 6 (Decidability of Type Checking). Given a Theory $\mathbb{T}$ and a type system $\vdash_{\mathcal{S}}$, it is decidable whether $\vdash_{\mathcal{S}} A : B$ for a given A, and B.

This property is probably true for the theory $\mathbb{T}_{\emptyset}$ and it heavily depends on the decidability of the theory $\mathbb{T}$ considered. In fact, the (*Conv*) type rule uses the relation $=_{\rho}$, which in turn depends on the theory $\mathbb{T}$ considered, which in turn depends on a suitable type derivation (see Definition 2.2). This sort of "circularity" might be a possible source of undecidability.

4.2 Conclusion

In this work we have defined a very powerful collection of nine (8+1) type systems whose main originality consists in using only one abstractor, namely the "$\to$" operator of the Rho Calculus, and matching as a primitive mechanism, leading thus to a behavior similar to the Π-reduction for types. We have enlighten with several examples the usefulness of such a calculus, its expressive power as well as its ability to avoid the drawbacks of the previous systems differentiating term-abstractors (*i.e.* λ) from type-abstractors (*i.e.* Π). The full proof-theoretical work for the ρ-cube is an ongoing work in progress.

This work opens many questions that we intend to solve now. They go from standard (and important) ones like proving subject reduction and strong normalization of typable terms, to extensions of the type system in order to take heterogeneous structures into account, and to the study of the more general setting of Pure Type Systems [Ber88,Ber90,Ter89]. We would also like to explore, following the Curry-Howard isomorphism, the possibility to define a "logical" ρ-cube.

Acknowledgement. The authors would like to thank Gilles Barthe and Fairouz Kamareddine for pointing out some crucial papers and the anonymous referees for the useful comments. We would also like to thank all the members of the ELAN group for their comments and interactions on the topics of the Rho Calculus.

References

Bar84. H. Barendregt. *Lambda Calculus: its Syntax and Semantics.* North Holland, 1984.

Bar92. H. Barendregt. Lambda Calculi with Types. In *Handbook of Logic in Computer Science*, volume II, pages 118–310. Oxford University Press, 1992.

Ber88. S. Berardi. Towards a Mathematical Analysis of Type Dependence in Coquand–Huet Calculus of Constructions and the Other Systems in Barendregt's Cube. Technical report, Dept. of Computer Science, Carnegie Mellon University, and Dip. di Matematica, Università di Torino, 1988.

Ber90. S. Berardi. *Type Dependence and Constructive Cathematics.* PhD thesis, Dipartimento di Matematica, Università di Torino, 1990.

Bür89. H.-J. Bürckert. Matching–A Special Case of Unification? *Journal of Symbolic Computation*, 8(5):523–536, 1989.

CH88. T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.

Chu41. A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1941.

Cir00. H. Cirstea. *Calcul de Réécriture : Fondements et Applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

CK99a. H. Cirstea and C. Kirchner. An Introduction to the Rewriting Calculus. Research Report RR-3818, INRIA, 1999.

CK99b. H. Cirstea and C. Kirchner. Combining Higher-Order and First-Order Computation Using ρ-calculus: Towards a Semantics of ELAN. In *Frontiers of Combining Systems 2*, pages 95–120. 1999.

CK00. H. Cirstea and C. Kirchner. The Typed Rewriting Calculus. In *Third International Workshop on Rewriting Logic and Application*, 2000.

CKL00. H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. Research Report RR-363, INRIA, 2000. Submitted.

Coq91. T. Coquand. Metamathematical Investigations of a Calculus of Constructions. In *Logic and Computer Science*, pages 91–122. Academic Press, 1991.

dB80. 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.

Gir72. J.Y. Girard. *Interpretation Fonctionnelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieur.* PhD thesis, Université Paris VII, 1972.

Gir86. J.Y. Girard. The System F of Variable Types, Fifteen Years Later. *Theoretical Computer Science*, 45:159–192, 1986.

GN91. H. Geuvers and M.J. Nederhof. A Modular Proof of Strong Normalization for the Calculus of Constructions. *J. of Functional Programming*, 1(2):155–189, 1991.

HHP92. R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *J. of the ACM*, 40(1):143–184, 1992.

Hul79. J.-M. Hullot. Associative-Commutative Pattern Matching. In *Proc. of IJCAI*, 1979.

JK91. J.-P. Jouannaud and C. Kirchner. Solving Equations in Abstract Algebras: A Rule-based Survey of Unification. In *Computational Logic. Essays in Honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, 1991.

KBN99. 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.

KK99. C. Kirchner and H. Kirchner. Rewriting, Solving, Proving. A preliminary version of a book available at `www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.

KL01. F. Kamareddine and T. Laan. A Correspondence between Martin-Löf Type Theory, the Ramified Theory of Types and Pure Type Systems. *J. Logic, Language and Information*, 2001. To appear.

KN96. F. Kamareddine and R. Nederpelt. Canonical Typing and π-conversion in the λ-cube. *J. Functional Programming*, 6(2):85–109, 1996.

Luo90. Z. Luo. ECC: An Extended Calculus of Constructions. In *Proceedings of LICS*, pages 385–395, 1990.

Mar84. P. Martin-Löf. *Intuitionistic Type Theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Naples, 1984.

PM97. S.L. Peyton Jones and E. Meijer. Henk: a Typed Intermediate Language. In *Types in Compilation Workshop*, 1997.

Ter89. J. Terlouw. Een nadere bewijstheoretische analyse van GSTTs. Manuscript, University of Nijmegen, Toernooiveld I, Nijmegen, The Netherlands, 1989.