Reasoning about Layered, Wildcard and Product Patterns

Delia Kesner

CNRS and LRI
Bât 490, Université de Paris-Sud
91405 Orsay Cedex
France
e-mail:Delia.Kesner@lri.fr

Abstract

We study the extensional version of the simply typed λ-calculus with product types and fixpoints enriched with *layered*, *wildcard* and *product patterns*. Extensionality is expressed by the surjective pairing axiom and a generalization of the η-conversion to patterns. We obtain a *confluent* reduction system by turning the extensional axioms as *expansion* rules, and then adding some restrictions to these expansions in order to avoid reduction loops. Confluence is proved by composition of modular properties of the extensional and non-extensional subsystems of the reduction calculus.

1 Introduction

Pattern-matching function definitions is one of the most popular features of functional languages, allowing to specify the behavior of functions by cases, according to the form of their arguments. Left-hand sides of function definitions are usually expressed using *Layered, Wildcard* and *Product Patterns* (LWPP), as for example the following Caml Light [ea93] program where the function `cons_new_pair` takes any pair (m_1, m_2) of type `'a * 'b` and yields the pair $(m_1, (m_1, m_2))$:

```
# let cons_new_pair =  function (y,_) as z -> (y,z) ;;

cons_new_pair : 'a * 'b -> 'a * ('a * 'b) = <fun>
```

The previous declaration can be written in the calculus we are studying in this paper, called the LWPP calculus, that is just the simply typed lambda calculus with product types and fixpoints, enriched with layered, wildcard and product patterns. These pattern constructors allow to make abstractions of functions not only with respect to variables, but also with respect to patterns, specifying in this way the form of the arguments of a function. Other formalisms for λ-calculi with nested patterns can be found in [vO90, Jon87, How92, BTKP93]. In particular, the LWPP calculus can also be viewed as a sub-calculus of the Typed Pattern Calculus (TPC) that was recently proposed in [BTKP93]. More precisely, the LWPP calculus can be encoded in the TPC following standard techniques: this last one is designed in the spirit of the Curry-Howard isomorphism as a computational interpretation of the Gentzen sequent proofs system for intuitionistic propositional logic, and so the application and projection constructors of the LWPP calculus are encoded in the TPC following the usual translation of natural deduction proofs into sequent proofs. See [Kes93] for details.

There is anyway something else that one likes to do with programs modeled by λ-calculi besides using λ-terms as programs to be computed: one would like to *reason* about programs, to *prove* that they enjoy certain properties and to *identify* expressions having the same meaning. For example, there are *at least* three ways to write the previous function `cons_new_pair` in the LWPP calculus:

Example 1.1

(1) $\lambda\langle x, y\rangle : A \times B.\langle x, \langle x, y\rangle\rangle$ (2) $\lambda x : A \times B.\langle \pi_1(x), x\rangle$ (3) $\lambda x @\langle y, \_\rangle : A \times B.\langle y, x\rangle$

Here is where conversion rules and extensional equalities come into play. In the case of the λ-calculus, there is deep correspondence between its axiomatic and operational semantics and the reduction rules may be derived by orienting each equational axiom. The choice to turn the equalities into rules has to be computationally reasonable and the reduction system has to be confluent. In this way the reduction rules play at the same time the rôle of a natural model of execution and a decision method for equational provability.

By far, the best known extensional equalities in the theory of typed λ-calculus are the (η) and (sp) axioms. They are essential to prove properties about pairs and functions.

The traditional (η) axiom for arrow types, is written in the λ-calculus formalism as follows:

$$(\eta) \qquad \lambda x : A.Mx = M \qquad \text{provided } x \text{ is not free in } M$$

Now, the existence of patterns in the scope of lambda abstractions suggests a natural extension of the (η)-axiom in order to express equalities between pattern abstractions [CCMS86]. The new axiom is called $(\widehat{\eta})$-axiom:

$$(\widehat{\eta}) \qquad \lambda P : A.MP = M \text{ provided } \begin{cases} P \text{ contains only variables and pairs} \\ \text{the variables of } P \text{ are not free in } M \end{cases}$$

On the other hand, there is also an extensional rule for pairs, called *surjective pairing*, which is the analog for product types of the usual (η)-axiom for arrow types, and can be specified by the equality:

$$(sp) \qquad \langle \pi_1(M), \pi_2(M)\rangle = M$$

where π_1 and π_2 are the first and second projection functions.

Note that the $\widehat{\eta}$-equality implies the categorical uniqueness of the product type as we can show $\langle fst\ z, snd\ z\rangle = z$, for $fst \equiv \lambda\langle x, y\rangle.x$ and $snd \equiv \lambda\langle x, y\rangle.y$. We refer the reader to [CCMS86] for details.

When one wants to turn these equalities into rules, there are two different choices. When turning them from left to right as *contractions*, they just state that a term $\langle \pi_1(M), \pi_2(M)\rangle$ is more complex than M and a term $\lambda x.Mx$ is more complex than M. Otherwise, the equalities can be turned as *expansions*, and they state that a term M of arrow type must really be a function, built by λ-abstraction, and a term M of product type has to be really a pair, built via the pair constructor.

The (η) and the (sp) axioms have traditionally been turned into contractions carrying the same name. Such an interpretation is well behaved in the simply typed λ-calculus as it preserves confluence [Pot81].

But surprisingly, recursion together with the (sp)-axiom oriented as contraction causes confluence to fail [Nes89]. Moreover, the contractive interpretation of the $(\widehat{\eta})$-axiom for patterns breaks confluence, as the following example shows:

Example 1.2

$$\lambda\langle x, y\rangle : A \times B.\langle x, y\rangle \xleftarrow{\beta} \lambda\langle x, y\rangle : A \times B.(\lambda z : A \times B.z)\langle x, y\rangle \xrightarrow{\widehat{\eta}} \lambda z : A \times B.z$$

Both terms $\lambda\langle x, y\rangle : A \times B.\langle x, y\rangle$ and $\lambda z : A \times B.z$ are in normal form and they are different, so the previous diagram cannot be closed.

This inconvenient can be fortunately overcome in the LWPP calculus by turning the extensional equalities into *expansion* rules. The idea was suggested by Prawitz [Pra71] in the field of proof theory, and further studied by Huet [Hue76] in the field of higher order unification, and by Mints [Min79] who was motivated by problems in category theory. But expansions seem however to have passed unnoticed for a long time, until these last two years, where many extensional λ-calculi (without patterns) have been investigated [Jay92, JG92, Cub92, Dou93, Aka93, DCK93a].

Mainly inspired by [Aka93, DCK93a], we use *expansion* rules to provide a *confluent* reduction system for the extensional typed pattern calculus with layered, wildcard and product patterns equipped with fixpoints and product types. In particular, our system allows to rewrite *any* function $\lambda P : A.M$ into another function $\lambda P' : A.M'$ such that the pattern P' contains only variables and pairs. This corresponds to the fact that wildcard and layered patterns are *redundant* but gives at the same time another interpretation: we can discard at compile time such redundant patterns (using our reduction system) as a form of optimization of programs that will be then evaluated by simpler computation rules.

Our reduction system may be regarded as an ideal (non-deterministic) implementation that allows to study arbitrary evaluation orders. We give however in the appendix A two evaluators, lazy and eager, and we refer the reader to [BTKP93] for more details about particular implementations of the Typed Pattern Calculus. We show that our reduction system is adequate for the axiomatic theory, *i.e,* that we do not loose equalities by the reduction relation. In particular, we can prove the three functions of example 1.1 to be equal since they have all the same normal form.

We show the confluence of the full system by a divide and conquer lemma that requires different properties: the confluence of the system without expansions (relation noted $\stackrel{B}{\Longrightarrow}$), the confluence and strong normalization of the system having only expansions (noted $\stackrel{E}{\Longrightarrow}$) and the property of *preservation of E-normal forms*: every time that a term M reduces to a term N in the system without expansions, then there is a non-empty $\stackrel{B}{\Longrightarrow}$-reduction sequence from the E-normal form of M to the E-normal form of N. This technique clarifies the relation between the reduction $\stackrel{B}{\Longrightarrow}$ and E-normalization and have also an independent interest since the explicit description of E-normal forms helps to understand the behavior of expansions in a pattern framework.

We introduce now the typing rules (section 2.1), the equalities (section 2.2) and the reduction system (section 2.3). We study the main properties of the reduction relations in section 3: confluence of $\stackrel{B}{\Longrightarrow}$ (section 3.1), strong normalization and confluence of $\stackrel{E}{\Longrightarrow}$ (section 3.2) and preservation of E-normal forms (section 3.3). We conclude with ideas for further work. Since we cannot go into all the details of the proofs by lack of space, we include as appendices the proofs that are mainly related to the expansionary interpretation of extensional rules in a pattern framework, and we refer the interested reader to [Kes93] for full details.

2 The Calculus

2.1 Terms and Types

It is now time to introduce the calculus LWPP. The set of *types* of our calculus is the least set containing a denumerable set of base types (as for example natural numbers and booleans), and closed w.r.t. formation of function and product types, *i.e,* if A and B are types, then also $A \rightarrow B$ and $A \times B$ are types.

In order to introduce the syntax of patterns and terms, we first fix a denumerable set of variables $\{x, y, z, \ldots\}$. The well-typed terms are given by the set of *typing judgements* in figure 1 which have the form $\Gamma \triangleright M : A$ where Γ, called a *pattern type assignment*, is a set of elements of the form $P : A$. Intuitively, the judgement $P_1 : A_1 \ldots P_n : A_n \triangleright M : A$ means that if patterns $P_1 \ldots P_n$ have types $A_1 \ldots A_n$ (respectively), then M is a well-typed term of type A.

Variables are typed by the following *axiom*:

$$x_1 : A_1,\ \ldots,\ x_n : A_n \triangleright x_i : A_i \quad (1 \leq i \leq n)$$

where the x_j's are pairwise distinct.

And terms are typed by the following *rules*:

Left rules :		
$(\times left)$	If $\Gamma,\ P_1 : A_1,\ P_2 : A_2 \triangleright M : B$,	then $\Gamma,\ \langle P_1, P_2\rangle : A_1 \times A_2 \triangleright M : B$
$(layered)$	If $\Gamma,\ P_1 : A,\ P_2 : A \triangleright M : B$,	then $\Gamma,\ P_1 @ P_2 : A \triangleright M : B$
$(wildcard)$	If $\Gamma \triangleright M : B$,	then $\Gamma,\ \_ : A \triangleright M : B$
Right rules :		
$(\times right)$	If $\Gamma \triangleright M_1 : A_1 \quad \Gamma \triangleright M_2 : A_2$,	then $\Gamma \triangleright \langle M_1, M_2\rangle : A_1 \times A_2$
(app)	If $\Gamma \triangleright M : A \rightarrow B \quad \Gamma \triangleright N : A$,	then $\Gamma \triangleright (MN) : B$
(abs)	If $\Gamma,\ P : A \triangleright M : B$,	then $\Gamma \triangleright \lambda P : A.M : A \rightarrow B$
$(proj_1)$	If $\Gamma \triangleright M : A \times B$,	then $\Gamma \triangleright \pi_1(M) : A$
$(proj_2)$	If $\Gamma \triangleright M : A \times B$,	then $\Gamma \triangleright \pi_2(M) : B$
$(recursion)$	If $\Gamma,\ x : A \triangleright M : A$,	then $\Gamma \triangleright \mu x : A.M : A$

Figure 1: Typing Rules for the LWPP Calculus

We use the left rules $(\times left)$, $(layered)$ and $(wildcard)$ to introduce patterns that correspond to devices long used in programming languages, and the right rules to introduce the terms of typed lambda calculus. The (abs) rule allows to abstract a function with respect to a pattern. The layered pattern $\_@\_$ introduced by the $(layered)$ rule is used to name in a unique way two different patterns of the same type and it corresponds to the construction *as* of ML, but here $P_1 @ P_2$ might contain any pattern P_1 and not just a variable. The pattern $\_$ introduced by the $(wildcard)$ rule is the usual wildcard pattern in ML and it denotes any possible argument, while the pattern $\langle P_1, P_2\rangle$ introduced by the $(\times left)$ rule is the usual product pattern (also called pair pattern).

We say that M is a well-typed term if there exists a pattern type assignment Γ and a type A such that $\Gamma \triangleright M : A$ is either an axiom or follows from axioms by the application of the typing rules. We use the notion of *context* for terms with "holes", assuming that holes do not appear in the patterns of λ-abstractions. We write $C[\,]$ for a context and we use $C[M]$ to denote the context $C[\,]$ where all the holes are replaced by M. We may omit types of patterns in $\lambda-$abstractions when they are clear from the context writing $\lambda P.M$ instead of $\lambda P : A.M$.

Free and bound occurrences of variables are defined as usual with the understanding that a term of the form $\lambda P : A.M$ defines bindings whose scope is M for all the variables occurring in P. A pattern is said to be *linear* if variables can occur at most once in it (note that all the patterns that can be constructed using the rules in figure 1 are linear). We denote by $Var(P)$

the set of variables that occurs in the pattern P, by $FV(M)$ the set of free variables that occurs in the term M, by $BV(M)$ the set of bound variables in the term M, and by $\mathcal{P}(M)$ the set of *positions* of the term M. We write $[N_1, \ldots, N_n/x_1, \ldots, x_n]$ (often abbreviated $[\overline{N}/\overline{x}]$) for the typed substitution mapping each variable $x_i : A_i$ to a term $N_i : A_i$ and $M[\overline{N}/\overline{x}]$ for the term M where each variable x_i free in M is replaced by N_i. In what follows, for every substitution θ, we assume $Dom(\theta) \cap \bigcup_{x \in Dom(\theta)} FV(\theta(x)) = \emptyset$.

As opposed to the simply typed λ-calculus, here a judgement may have several derivations. However, this is not so problematic as the left rules $(layered)$, $(wildcard)$ and $(\times left)$ commute with all the other rules - they have the so-called *commutation property* - and therefore, for every *non-variable* term M there exists *exactly one* rule such that every judgement $\Gamma \triangleright M : E$ has a derivation ending with the application of that rule. Indeed, for any judgement of a pair $\langle M, N\rangle$ there is a derivation ending with the $(\times left)$ rule, for any judgement of an abstraction $\lambda P : A.M$ there is a derivation ending with the (abs) rule, in the case of an application (MN) the associated rule is (app), for the recursion term $\mu x : A.M$ the associated rule is $(recursion)$, and finally the $(proj_i)$ rule corresponds to the terms of the form $\pi_i(M)$.

We only use patterns built from variables and the pair constructor in order to define the extensional equalities. For that, we introduce the notion of *specification* and *full specification* of a pattern P by a type A, writing $P :\cdot A$ and $P :: A$ respectively. The notion of specification avoids the association of non product types to pair patterns, and the notion of full specification, besides that, forbid the association of product types to variables, *i.e,* a full specified pattern has a product type if and only if it is a pair pattern. A similar notion is defined for terms.

Definition 2.1 (Specification of Patterns) For every variable x and every type A we have $x :\cdot A$ and, whenever $P_1 :\cdot A_1$ and $P_2 :\cdot A_2$, then $\langle P_1, P_2\rangle :\cdot A_1 \times A_2$.

Definition 2.2 (Full Specification of Patterns) For every variable x and every non product type A we have $x :: A$ and, whenever $P_1 :: A_1$ and $P_2 :: A_2$, then $\langle P_1, P_2\rangle :: A_1 \times A_2$.

Definition 2.3 (Full Specification of Terms) A term M is *fully specified* if M is fully specified by its type, and M is *fully specified* by a type A if and only if M is of type A and they correspond to one of the following cases:

- If $M \neq \langle M_1, M_2\rangle$ and $A \neq A_1 \times A_2$, then $M :: A$
- If $M_1 :: A_1$ and $M_2 :: A_2$, then $\langle M_1, M_2\rangle :: A_1 \times A_2$.

Example 2.4 Let A and B be two base type, z and $\langle x, y\rangle$ be two patterns, and $\pi_1(w)$ and $\langle \pi_1(w), z\rangle$ two terms. Then $z :\cdot A \times B$ and $\langle x, y\rangle :\cdot A \times B$ but $z \not{::} A \times B$ and $\langle x, y\rangle \not{:\cdot} A$. In the case of terms, suppose that w is of type $A \times B$ and z is of type B. Then $\pi_1(w) :: A$ and $\langle \pi_1(w), z\rangle :: A \times B$, but $w \not{::} A \times B$ and $\langle w, z\rangle \not{::} A$.

It is clear that a fully specified pattern is also a specified pattern and whenever a term M is fully specified by a type A and σ is a well-typed substitution, then $N[\sigma]$ is also fully specified by A.

2.2 Equalities

First of all we define a function $Match(M, P)$ on every term M and every pattern P that allows to generalize the usual β rule to the case of patterns. This function explains clearly the operational semantics of layered, wildcard and product patterns by the notion of matching substitution, which is defined as follows:

$$
\begin{array}{lll}
Match(N, \_) & = \emptyset & \\
Match(N, x) & = [N/x] & \\
Match(N, P_1 @ P_2) & = Match(N, P_1) \cup Match(N, P_2) & \\
Match(\langle N_1, N_2\rangle, \langle P_1, P_2\rangle) & = Match(N_1, P_1) \cup Match(N_2, P_2) & \\
Match(M, P) & = \mathbf{fail} & \text{Otherwise}
\end{array}
$$

Note that $Match(\_,\_)$ is well-defined in terms of the operation $\cup$ because all patterns are linear according to the typing rules in figure 1. Using this definition we can show the existence of matching substitutions between fully specified terms and specified patterns, and also, that the substitution lemma holds for our calculus.

Lemma 2.5 *If P is a pattern such that $P :\cdot A$ and N is a term such that $N :: A$, then there is a substitution σ such that $Match(N, P) = \sigma$ and $\sigma(P) = N$.*

Lemma 2.6 (Substitution Lemma)
If $\Gamma, P : C \triangleright M : A$, $\Gamma \triangleright N : C$ and $Match(N, P) = \theta$, then $\Gamma \triangleright M[\theta] : A$.

We are now able to define the equalities between terms of our calculus. The α-conversion is the most natural one, it allows to rename bound variables. If ψ is a renaming substitution, the α-equality for λ-abstractions corresponds to the equation:

$$
(\alpha) \qquad \lambda P : A.M =_\alpha \lambda P[\psi] : A.M[\psi], \text{ where } \left\{ \begin{array}{l} Var(P[\psi]) \cap (FV(M) - Var(P)) = \emptyset \\ Var(P[\psi]) \cap BV(M) = \emptyset \end{array} \right.
$$

For example, $\lambda\langle x, y\rangle.x =_\alpha \lambda\langle z, w\rangle.z$ and $\lambda x@y.\langle x, y\rangle =_\alpha \lambda y@x.\langle y, x\rangle$.
Besides α conversion, our calculus is equipped with the following equalities between terms:

$$
\begin{array}{llll}
(\beta) & (\lambda P : A.M)N & =_\beta & M[\theta], \text{ If } Match(N, P) = \theta \\
(@) & (\lambda P_1 @ P_2 : A.M)N & =_@ & (\lambda\langle P_1, P_2\rangle : A \times A.M)\langle N, N\rangle \\
(\pi_i) & \pi_i\langle M_1, M_2\rangle & =_{\pi_i} & M_i, \text{ for } i = 1, 2 \\
(rec) & \mu x : C.M & =_{rec} & M[\mu x : C.M/x]
\end{array}
$$

Axiom ($@$) tells us that both terms $(\lambda P_1 @ P_2 : A.M)N$ and $(\lambda\langle P_1, P_2\rangle : A\times A.M)\langle N, N\rangle$ have the same behavior (since $Match(N, P_1 @ P_2)$ and $Match(\langle N, N\rangle, \langle P_1, P_2\rangle)$ yield the same substitution), and therefore, we can translate the first into the second one. However, the term $(\lambda P_1 @ P_2 : A.M)$ is completely different from $(\lambda\langle P_1, P_2\rangle : A \times A.M)$: the first one is an abstraction on just one argument that is denoted in two different ways, whereas the second one is an abstraction on two different arguments.

We define the conversion $=_{LWPP}$ as the least congruence containing $=_\alpha$, $=_\beta$, $=_@$, $=_{\pi_1}$, $=_{\pi_2}$ and $=_{rec}$.

Definition 2.7 (The LWPP calculus) The *LWPP calculus* is equipped with the typing rules in figure 1 and the conversion $=_{LWPP}$.

The following conversion axioms are the extensional rules of the LWPP calculus, where the axiom $\widehat{\eta}$ is the natural extension of the η-equality in the simply typed λ-calculus.

$$(\widehat{\eta}) \qquad M =_{\widehat{\eta}} \lambda P : A.(MP) \quad \text{If} \begin{cases} Var(P) \cap FV(M) = \emptyset \\ M \text{ is of type } A \to B \\ P :\cdot A \end{cases}$$

$$(sp) \qquad M =_{sp} \langle \pi_1(M), \pi_2(M) \rangle \text{ If } M \text{ is of type } A \times B$$

Definition 2.8 (The extensional calculus LWPP) The *extensional calculus LWPP* contains the typing rules in figure 1 and the conversion $\doteq$ defined as the $=_{LWPP}$ equality enriched with $=_{\widehat{\eta}}$ and $=_{sp}$.

2.3 The reduction system

While equations correspond to the axiomatic semantics of the calculus, reduction rules give the operational semantics as a form of symbolic computation, *i.e,* it models program execution. For that, a reduction system has to verify some general properties, as for example *confluence* and *uniqueness of normal forms.* It is well-known that the first property guarantees the second one, so we are left to insure confluence by means of an appropriate reduction system.

Traditionally, a reduction system is obtained from the equational one by simply orienting the equations in a convenient way. But this is not the case for our calculus, where a reduction system with good properties needs to be defined in a more delicate way.

In order to obtain a reduction system for the LWPP calculus we first turn the *non extensional rules* from left to right in the following way:

Definition 2.9 (The reduction relation $\stackrel{\beta,@,\pi_1,\pi_2,rec}{\longrightarrow}$)

$$\begin{array}{lrcl}
(\beta) & (\lambda P : A.M)N & \longrightarrow & M[\theta], \text{ If } Match(N,P) = \theta \\
(@) & (\lambda P_1 @ P_2 : A.M)N & \longrightarrow & (\lambda\langle P_1, P_2\rangle : A \times A.M)\langle N, N\rangle \\
(\pi_i) & \pi_i(\langle M_1, M_2\rangle) & \longrightarrow & M_i \ (i = 1, 2) \\
(rec) & \mu x : C.M & \longrightarrow & M[\mu x : C.M/y]
\end{array}$$

When turning the *extensional equalities* into reduction rules, the contractive choice breaks confluence (example 1.2) while the expansive choice breaks strong normalization (example 2.10).

Example 2.10

$$\langle M, N\rangle \rightsquigarrow \langle \pi_1(\langle M, N\rangle), \pi_2(\langle M, N\rangle)\rangle \rightsquigarrow \langle M, \pi_2(\langle M, N\rangle)\rangle \rightsquigarrow \langle M, N\rangle$$

Even if recursion gives a natural mechanism to produce non terminating sequences, we prefer to keep expansions as strongly normalizing rules to provide a characterization of $(\widehat{\eta}, sp)$-normal forms that are usually very useful when working with η-long normal forms in higher order unification. For that, strong normalization for extensional rules is recovered when some restrictions are imposed to them. In order to define an appropriate reduction system for our calculus let us analyze in detail all the loops generated by the arbitrary application of extensional rules turned as expansions. There are two different situations in which such loops may arise, we call them *structural loops* and *contextual loops* respectively. Structural loops are caused by the application of an expansion rule to a term having a *structure* that is sufficiently specified, while contextual loops are caused by the application of an expansion rule to a subterm that appears in a particular *context.* Let us see all that in detail.

Structural Loops

The structural loop caused by *sp* expansions appears in the example 2.10 and can be avoided by forbidding expansions on terms that are yet pairs as follows:

Definition 2.11 (The reduction relation $\xrightarrow{sp}$)

$$(sp)\ \ M \xrightarrow{sp} \langle \pi_1(M), \pi_2(M)\rangle,\ \text{If} \begin{cases} M \text{ is of type } A \times B \\ M \text{ is not a pair} \end{cases}$$

Let see now the loops caused by the application of the $\widehat{\eta}$ expansion, and in particular let us start by a case that is also problematic in the typed λ-calculus:

$$\lambda x : A.M \leadsto \lambda x' : A.(\lambda x : A.M)x' \leadsto \lambda x' : A.M[x'/x] =_\alpha \lambda x : A.M$$

We can reproduce this situation in the presence of more complicated patterns, where the loop starts in a term having a pair as abstraction.

$$\begin{array}{l} \lambda\langle x, y\rangle : A \times B.M \leadsto \lambda\langle x', y'\rangle : A \times B.(\lambda\langle x, y\rangle : A \times B.M)\langle x', y'\rangle \leadsto \\ \lambda\langle x', y'\rangle : A \times B.M[x'/x][y'/y] =_\alpha \lambda\langle x, y\rangle : A \times B.M \end{array}$$

In general, each time M is a λ-abstraction $\lambda Q : A.N$, and it reduces via the $\widehat{\eta}$-rule to a term $\lambda P : A.(\lambda Q : A.N)P$, where the pattern P is "as specific as" Q (*i.e,* α-equivalent to Q), we can always perform a β step to close the loop. As a consequence, whenever an $\widehat{\eta}$ expansion is applied to a λ-abstraction $\lambda Q : A.N$, it has to introduce a pattern P strictly "less specific" or "more specific" than Q. This suggests that it is necessary to chose a canonical form of applying an $\widehat{\eta}$ expansion to λ-abstractions : either we always introduce less specific patterns, or we always use more specific ones.

But the interpretation of extensional equalities as expansions have the intention to capture as much information as possible: if a term is of product type, then it is a pair, and if a term is of functional type, then it is a λ-abstraction. It is then reasonable, according to our interpretation, to chose the largest patterns, and it is now that the notions of *specification* and *full specification* come into play. The general form of the $\widehat{\eta}$-rule is:

Definition 2.12 (The reduction relation $\xrightarrow{\widehat{\eta}}$)

$$(\widehat{\eta})\ \ M \xrightarrow{\widehat{\eta}} \lambda P : A.(MP),\ \text{If} \begin{cases} Var(P) \cap FV(M) = \emptyset \\ P :: A \\ M \text{ is of type type } A \to B \\ M = \lambda Q : A.N \text{ implies } Q \not:: A \end{cases}$$

We can immediately verify that all the loops that we have previously shown are now ruled out.

Contextual Loops

Now, let us analyze the contextual loops, that are produced by the application of expansions to subterms that are placed in particular positions of a term. For example, in the case of the *sp*-rule, a loop arises when applying the *sp* expansion to a term M that is the argument of a projection:

$$\pi_1(M) \leadsto \pi_1(\langle \pi_1(M), \pi_2(M)\rangle) \leadsto \pi_1(M)$$

In order to avoid this kind of loops, the *sp* expansions will not be allowed on terms that are projected.

We can also produce some contextual loops by applying an $\widehat{\eta}$ expansion to a term M that is applied to another term N of base type: indeed, $MN \leadsto (\lambda x.Mx)N \leadsto MN$.

In the presence of patterns the situation is more complex as the application of the $\widehat{\eta}$ expansion does not always cause a loop. Suppose for example that A and B are base types and let N be a term of type $A \times B$ but different from a pair. Then, $MN \leadsto (\lambda\langle z, w\rangle : A \times B.M\langle z, w\rangle)N$. However, if we take N to be a pair $\langle N_1, N_2\rangle$ we obtain:

$$M\langle N_1, N_2\rangle \leadsto (\lambda\langle z, w\rangle : A \times B.M\langle z, w\rangle)\langle N_1, N_2\rangle \leadsto M\langle N_1, N_2\rangle$$

In general, this situation corresponds to the case of a term M applied to another term L that is fully specified by its type (in the example $\langle N_1, N_2\rangle$ is fully specified by the type $A \times B$), and thus the term $\lambda P.MP$ coming from the application of the $(\widehat{\eta})$-rule to the term M, produces by lemma 2.5 a matching substitution θ such that $\theta(P) = L$. The loops starting at ML are of the form:

$$ML \leadsto (\lambda P.MP)L \leadsto ML$$

To break this kind of loops, we will also forbid an $\widehat{\eta}$ expansion on a subterm that is applied to a fully specified term.

In summary, the one-step reduction relation $\Longrightarrow$ cannot be defined as the least congruence on terms containing the relation $\longrightarrow$ above, as is done traditionally. We proceed as follows:

Definition 2.13 (Relation $\Longrightarrow$) The one-step reduction relation between terms, denoted $\Longrightarrow$, is defined to be the closure of the reduction rules $\beta, @, \pi_1, \pi_2, rec, \widehat{\eta}, sp$ for all the contexts *except* application and projection, *i.e*:

- If $M \Longrightarrow M'$, then $MN \Longrightarrow M'N$ except in the case where $M \xrightarrow{\eta} M'$ and N is fully specified.
- If $M \Longrightarrow M'$, then $\pi_i(M) \Longrightarrow \pi_i(M')$ for $i = 1, 2$ except in the case where $M \xrightarrow{sp} M'$

Notation 2.1 The transitive and the reflexive transitive closure of $\Longrightarrow$ are noted $\Longrightarrow^+$ and $\Longrightarrow^*$ respectively.

We will use some standard notions from the theory of rewriting system, such as redex, normal form, confluence, weak confluence, strong normalization, etc, without explicitly redefining them here.

Using lemma 2.6 and the commutation property, it is easy to show that the one-step reduction relation preserves types.

Theorem 2.14 (Subject Reduction) *If* $\Gamma \triangleright M : A$ *and* $M \Longrightarrow M'$, *then* $\Gamma \triangleright M' : A$

The following theorem states the redundancy of layered and wildcard patterns by means of our reduction system. Indeed, every pattern abstraction $\lambda P : A.M$ can be rewritten to a function $\lambda P' : A.M'$ such that P' contains only variables and pairs, and so we can show the following:

Theorem 2.15 (Redundancy of Layered and Wildcard Patterns) *Any term M can be reduced to a term M' which contains no layered and no wildcard patterns.*

Proof. See Appendix B.

$\square$

Coming back to example 1.1, we can check the three terms to verify the property stated in theorem 2.15. Indeed, the property is trivial for $\lambda\langle x, y\rangle : A \times B.\langle x, \langle x, y\rangle\rangle$ and $\lambda x : A \times B.\langle \pi_1(x), x\rangle$ because they have no layered and no wildcard patterns, while for $\lambda x@\langle y, \_\rangle : A \times B.\langle y, x\rangle$ we have to perform the following reduction sequence:

$$\begin{array}{ll}
\lambda x@\langle y, _\rangle : A \times B.\langle y, x\rangle & \stackrel{\widehat{\eta}}{\Longrightarrow} \\
\lambda\langle w, z\rangle : A \times B.(\lambda x@\langle y, _\rangle : A \times B.\langle y, x\rangle)\langle w, z\rangle & \stackrel{@}{\Longrightarrow} \\
\lambda\langle w, z\rangle : A \times B.(\lambda\langle x, \langle y, _\rangle\rangle : (A \times B) \times (A \times B).\langle y, x\rangle)\langle\langle w, z\rangle, \langle w, z\rangle\rangle & \stackrel{\beta}{\Longrightarrow} \\
\lambda\langle w, z\rangle : A \times B.\langle w, \langle w, z\rangle\rangle &
\end{array}$$

The adequacy of expansions for extensional equalities comes from the following theorem, which states that the limitations imposed on the reduction system do not make us loose any valid equality as they are introduced exactly to avoid reduction loops.

Theorem 2.16 ($\Longrightarrow$ generates $\doteq$) *$M \doteq N$ if and only if $M \stackrel{*}{\Longleftrightarrow} N$.*

Proof. See Appendix B.

$\square$

Coming back again to example 1.1, the three terms can be proved to be equal using the reduction relation $\Longrightarrow$ as follows:

$$\begin{array}{llll}
\lambda x : A \times B.\langle \pi_1(x), x\rangle & \stackrel{\widehat{\eta}}{\Longrightarrow} & \lambda\langle w, z\rangle : A \times B.(\lambda x : A \times B.\langle \pi_1(x), x\rangle)\langle w, z\rangle & \stackrel{\beta}{\Longrightarrow} \\
\lambda\langle w, z\rangle : A \times B.\langle \pi_1(\langle w, z\rangle), \langle w, z\rangle\rangle & \stackrel{\pi_1}{\Longrightarrow} & \lambda\langle w, z\rangle : A \times B.\langle w, \langle w, z\rangle\rangle &
\end{array}$$

and as previously shown:

$$\lambda x@\langle y, \_\rangle : A \times B.\langle y, x\rangle \Longrightarrow^* \lambda\langle w, z\rangle : A \times B.\langle w, \langle w, z\rangle\rangle$$

3 Confluence of the extensional LWPP calculus

We have now to show that the relation $\Longrightarrow$ is confluent. Standard techniques for showing confluence of the typed λ-calculus will not work directly, because of the extensional expansions. To overcome this annoyance, we use a divide-and-conquer technique as in [Aka93]: we separate the reduction $\Longrightarrow$ in two reduction relations: $\stackrel{B}{\Longrightarrow}$ denotes the relation $\Longrightarrow$ generated only by $\stackrel{\beta,\pi_1,\pi_2,@,rec}{\longrightarrow}$ and $\stackrel{E}{\Longrightarrow}$ denotes the relation $\Longrightarrow$ generated only by $\stackrel{\widehat{\eta},sp}{\longrightarrow}$. The confluence property of $\Longrightarrow$ comes from the following proposition:

Proposition 3.1 *If*

1. *$\stackrel{B}{\Longrightarrow}$ is confluent*
2. *$\stackrel{E}{\Longrightarrow}$ is confluent and strongly normalizing*
3. *$\forall M, N \;\; M \stackrel{B}{\Longrightarrow} N$ implies $M^E \stackrel{B}{\Longrightarrow}{}^+ N^E$, where M^E and N^E are respectively the E-normal forms of M and N*

then the relation $\stackrel{E}{\Longrightarrow} \cup \stackrel{B}{\Longrightarrow}$ is confluent.

Proof. See for example [Har87].

$\square$

Next sections discuss the techniques to prove the three properties required by the previous lemma: confluence of the relation $\stackrel{B}{\Longrightarrow}$, confluence and strong normalization of the relation $\stackrel{E}{\Longrightarrow}$, and the third property called *preservation of E-normal forms.*

3.1 Confluence of $\stackrel{B}{\Longrightarrow}$

The confluence property of $\stackrel{B}{\Longrightarrow}$ is shown by the Tait-Martin Löf's technique. The first step is to define a reduction relation $\blacktriangleright$ in order to capture *parallel* reduction, then to check that $\blacktriangleright$ enjoys the diamond property (which immediately implies confluence) and finally, to relate the reflexive-transitive closures of the relations $\blacktriangleright$ and $\stackrel{B}{\Longrightarrow}$. We start by defining the relation $\blacktriangleright$:

- $M \blacktriangleright M$
- If $M \blacktriangleright M'$, then $\pi_1(\langle M, N\rangle) \blacktriangleright M'$, $\pi_2(\langle N, M\rangle) \blacktriangleright M'$, $\mu x : A.M \blacktriangleright M'[\mu x : A.M'/x]$ and $C[M] \blacktriangleright C[M']$, where $C[\,]$ is a context $\lambda P.[\,]$, $\mu x : A.[\,]$ or $\pi_i([\,])$.
- If $M \blacktriangleright M'$ and $N \blacktriangleright N'$, then $\langle M, N\rangle \blacktriangleright \langle M', N'\rangle$, $MN \blacktriangleright M'N'$ and $(\lambda P : A.N)M \blacktriangleright N'[Match(M', P)]$ if $Match(M, P)$ is defined
- If $M \blacktriangleright M'$, $M \blacktriangleright M''$ and $N \blacktriangleright N'$, then $(\lambda P_1 @ P_2 : A.N)M \blacktriangleright (\lambda\langle P_1, P_2\rangle : A\times A.N')\langle M', M''\rangle$

The following lemma states that parallel reduction is stable by substitution and it plays an important rôle in the confluence proof.

Lemma 3.2 *If $M \blacktriangleright M'$, $N \blacktriangleright N'$ and $Match(N, P)$ is defined, then* $M[Match(N, P)] \blacktriangleright M'[Match(N', P)]$

Theorem 3.3 *The reduction relation $\stackrel{B}{\Longrightarrow}$ is confluent.*

Proof. By Tait-Martin Löf's technique: we first show the following three properties:

- The relation $\blacktriangleright$ satisfies the the diamond property (use lemma 3.2).
- A relation satisfying the diamond property is confluent.
- The reflexive transitive closures of $\blacktriangleright$ and $\stackrel{B}{\Longrightarrow}$ are the same relation.

We can now derive the confluence property of the relation $\stackrel{B}{\Longrightarrow}$ in terms of the confluence property of the relation $\blacktriangleright$.

$\square$

3.2 Strong Normalization and Confluence of $\stackrel{E}{\Longrightarrow}$

We show in this section that the reduction relation $\stackrel{E}{\Longrightarrow}$ is strongly normalizing and confluent. For the strong normalization property, we adapt the proof of [Min79] for the typed λ-calculus. We assign to every term T a natural number $E(T)$ and we prove that $T \stackrel{E}{\Longrightarrow} T'$ implies $E(T) > E(T')$, which shows that $E(T)$ is an upper bound for the length of any $\stackrel{E}{\Longrightarrow}$-reduction sequence. We note $|A|$ the number of operations and base types in the type A.
For every $u \in \mathcal{P}(T)$:

$$D_T(u) = \begin{cases} 2^{\Sigma_{1\leq i\leq n}|A_i|} & \text{If } T/u \text{ is an } E\text{-redex} \\ 1 & \text{Otherwise} \end{cases}$$

where $A_1 \ldots A_n$ are the types of E-redexes in T that contain T/u (even T/u and T)

For each $u \in \mathcal{P}(T)$ such that $u \neq \epsilon$:

$$C_T(u) = \begin{cases} 2^{\Sigma_{1\leq i\leq n}|A_i|} & \text{If } T/u \text{ is an } E\text{-redex} \\ 1 & \text{Otherwise} \end{cases}$$

where $A_1 \ldots A_n$ are the types of E-redexes in T that are different from T and contain T/u.

To each term T we associate the natural number $E(T)$:

$$E(T) = \Sigma_{u\in\mathcal{RP}(T)} D_T(u), \text{ where } \mathcal{RP}(T) \text{ is the set of } E\text{-redexes positions in } T.$$

Theorem 3.4 *The relation $\stackrel{E}{\Longrightarrow}$ is strongly normalizing.*

Proof. We show that $E(M) > E(M')$ if $M \stackrel{E}{\Longrightarrow} M'$ by case-analysis and then induction on the structure of terms. See [Kes93] for details.

□

Theorem 3.5 *The relation $\stackrel{E}{\Longrightarrow}$ is confluent.*

Proof. Is a consequence of theorem 3.4 and Newman's lemma because the relation $\stackrel{E}{\Longrightarrow}$ is easily checked to be weakly confluent.

□

3.3 Characterization and Preservation of E-normal forms

An important property of a one-step expansion $M \xrightarrow{\widehat{\eta}} M'$ or $M \xrightarrow{sp} M'$ is that the term M appears as a subterm of the term M', *i.e,* M' is obtained by inserting M in a special *expansor context*[1], which represents the scheme of the expansion rules $\xrightarrow{\widehat{\eta}}$ and $\xrightarrow{sp}$. Let us see that on a simple example:

Example 3.6 Let z be a variable of type $A_1 \times A_2$, where the A_i's are base types. Applying a *sp*-expansion to z, we obtain its E-normal form $\langle \pi_1(z), \pi_2(z)\rangle$, that can be obtained by inserting the variable z in the holes of the context $\langle \pi_1([\,]), \pi_2([\,])\rangle$.

In the general case, there are many positions of a term M where an expansion can be performed, so the characterization of its E-normal form have to take into account the *structure* of M. In the previous example 3.6, to define the E-normal form of the term $\langle z, z\rangle$, each occurrence of the variable z has to be inserted in the holes of an expansor context but the pair itself does not. This is due to the fact that each occurrence of z in $\langle z, z\rangle$ is reducible by $\stackrel{sp}{\Longrightarrow}$ but the pair $\langle z, z\rangle$ is not reducible at the root position.

[1] To differentiate with the expansor *terms* defined in [DCK93a]

On the other hand, expansor contexts must be carefully defined to correctly represent not only the expansion step arising from a redex already present in M, but also all the expansion sequences that such step can create: if in the previous example A_1 is taken to be an arrow type and A_2 a product type, then the term $\pi_1(z)$ can be further $\widehat{\eta}$-expanded and the term $\pi_2(z)$ can be expanded by the *sp*-expansion. However, the expansor context $\langle \pi_1([\,]), \pi_2([\,])\rangle$ cannot simulate these further possible reductions. This can only be done by storing in the expansor contexts all the information on possible future expansions, that is fully contained in the *type* of the term we are marking.

Definition 3.7 (Expansor Contexts) We associate to every type C an *expansor context* $\Delta_C[\,]$, defined by induction on the size of C as follows:

$$\begin{array}{lll} \Delta_{A\to B}[\,] & = & \lambda P : A.\Delta_B[[\,]\widehat{P}], \text{ where } P :: A \text{ and } \widehat{P} = \begin{cases} \langle \widehat{P_1}, \widehat{P_2}\rangle & \text{If } P = \langle P_1, P_2\rangle \\ \Delta_A[P] & \text{Otherwise} \end{cases} \\ \Delta_{A\times B}[\,] & = & \langle \Delta_A[\pi_1[\,]], \Delta_B[\pi_2[\,]]\rangle \\ \Delta_A[\,] & & \text{is the empty context in any other case} \end{array}$$

We write $\Delta_A?[M]$ to denote either $\Delta_A[M]$ or M.

Definition 3.8 (Translation) For every term M of type A, its *translation* M° is defined as follows:

$$M^\circ = \begin{cases} M^{\circ\circ} & \text{If } \begin{cases} M = \langle P_1, P_2\rangle \text{ or} \\ M = \lambda P : B.N \text{ and } P :: B \end{cases} \\ \Delta_A[M^{\circ\circ}] & \text{otherwise} \end{cases}$$

and $M^{\circ\circ}$ is defined by induction on the structure of M as:

$$\begin{array}{lll} x^{\circ\circ} = x & \pi_i(M)^{\circ\circ} = \pi_i(M^{\circ\circ}) & (\lambda P : B.M)^{\circ\circ} = \lambda P : B.M^\circ \\ \langle M, N\rangle^{\circ\circ} = \langle M^\circ, N^\circ\rangle & (MN)^{\circ\circ} = (M^{\circ\circ}N^\circ) & \mu y : A.M^{\circ\circ} = \mu y : A.M^\circ \end{array}$$

Example 3.9 Let z_1 be of type $(A \times A') \to B$ and z_2 be of type $C \times D$, where A, A', B, C and D are base types. We have $\langle z_1, z_2\rangle^\circ = \langle z_1, z_2\rangle^{\circ\circ} = \langle z_1^\circ, z_2^\circ\rangle = \langle \Delta_{(A\times A')\to B}[z_1], \Delta_{C\times D}[z_2]\rangle$, where

$$\Delta_{(A\times A')\to B}[z_1] = \lambda\langle x, y\rangle : A \times A'.z_1\langle x, y\rangle \quad \text{and} \quad \Delta_{C\times D}[z_2] = \langle \pi_1(z_2), \pi_2(z_2)\rangle$$

It is quite easy to proof that the translation preserves types and characterizes E-normal forms, so we can now complete the proof of confluence of $\Longrightarrow$ by showing that for every one-step reduction $\overset{B}{\Longrightarrow}$ from M to N, there is a non-empty $\overset{B}{\Longrightarrow}$-reduction sequence from the E-normal form of M to the E-normal form of N.

Theorem 3.10 (Preservation of E-normal forms) *If* $\Gamma \rhd M : A$ *and* $M \overset{B}{\Longrightarrow} N$, *then* $M^\circ \overset{B}{\Longrightarrow}{}^{+} N^\circ$.

Proof. By induction on the structure of M using the fact that $\Delta_A?[M^\circ[\overline{N^\circ}/\overline{x}]] \overset{\beta,\pi_i}{\Longrightarrow}{}^{*} (M[\overline{N}/\overline{x}])^\circ$. □

Theorem 3.11 (Confluence of $\Longrightarrow$) *The relation* $\Longrightarrow$ *is confluent.*

Proof. Using proposition 3.1 and theorems 3.3, 3.4, 3.5 and 3.10. □

The confluence property guarantees uniqueness of normal forms, so that results are independent of the evaluation strategy. This is a minimal requirement for our calculus to model program execution and is essential for parallel implementations of functional languages.

4 Conclusion and Future Work

We have provided a confluent reduction system for the extensional typed λ-calculus with wildcard, layered and product patterns equipped with fixpoints and product types.

We have shown that even if wildcard and layered patterns allow more friendly specifications of function definitions (as example 1.1 shows), they are redundant and can be eliminated. That is, for *any* function $\lambda P : A.M$, there is another function $\lambda P' : A.M'$ equal to $\lambda P : A.M$ modulo our equational theory such that the pattern P' only contain variables and product patterns.

This is not very surprising because it corresponds to the fact that contraction and weakening rules are redundant in intuitionist logic [Gal93] and gives at the same time another possible reading: to remove at compile time wildcard and layered patterns from λ-abstractions using our reduction system, could lead to optimizations of programs which will be then interpreted by simpler computation rules. In fact, the (@) rule does not make more sense and the pattern matching function $Match(\_, \_)$ become simpler.

The reduction system for the LWPP calculus without recursion can also be used as a decision method for equational provability as a consequence of theorem 2.16 and theorem 3.11. This is clearly illustrated by the example at the end of section 2.3.

This work clarifies the relation between extensionality and layered, wildcard and product patterns, but it rests as future work the development of an axiomatic theory for the Typed Pattern Calculus, especially in the presence of patterns of sum type. It is also important to remark that this approach can be applied to many other calculi with expansionary rules: for example, the weak extensionality for the sum type as in [DCK93b] or combinations of algebraic rewriting systems with extensional lambda calculi [DCK94]. We also believe that polymorphic calculi like Girard's Systems F can be handled within an expansionary interpretation of extensional rules.

Acknowledgements

I would like to thank Roberto Di Cosmo for the valuable comments made on an earlier version of this paper and for his careful proofreading. I am also very grateful to Val Breazu-Tannen for stimulating and helpful discussions about various topics related to this work.

References

[Aka93] Yohji Akama. On mints' reductions for ccc-calculus. In *Typed Lambda Calculus and Applications*, number 664 in LNCS. Springer Verlag, 1993.

[BTKP93] Val Breazu-Tannen, Delia Kesner, and Laurence Puel. A typed pattern calculus. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, Montreal, Canada, June 1993.

[CCMS86] Guy Cousineau, Pierre-Louis Curien, Michel Mauny, and Ascánder Suárez. Combinateurs catégoriques et implémentation des langages fonctionnels. Technical Report LIENS-86-3, Laboratoire d'Informatique du Département de Mathématiques et d'Informatique de l'École Normale Supérieur, 1986.

[Cub92] Djordje Cubric. On free ccc. Distributed on the `types` mailing list, 1992.

[DCK93a] Roberto Di Cosmo and Delia Kesner. A confluent reduction for the extensional typed $\lambda-$calculus with pairs, sums, recursion and terminal object. In Andrzej Lingas, editor, *Intern. Conf. on Automata, Languages and Programming (ICALP)*, number 700 in LNCS. Springer-Verlag, 1993.

[DCK93b] Roberto Di Cosmo and Delia Kesner. Simulating expansions without expansions. *Mathematical Structures in Computer Science*, 1993. To appear. Available also as INRIA Technical Report 1911.

[DCK94] Roberto Di Cosmo and Delia Kesner. Combining first order algebraic rewriting systems, recursion and extensional typed lambda calculi. In *Intern. Conf. on Automata, Languages and Programming (ICALP)*, LNCS. Springer-Verlag, 1994. To appear.

[Dou93] Daniel Dougherty. Some lambda calculi with categorical sums and products. In *Proc. of the Fifth International Conference on Rewriting Techniques and Applications (RTA)*, 1993.

[ea93] Xavier Leroy et al. The Caml Light system, release 0.6. Software and documentation distributed by anonymous FTP on `ftp.inria.fr`, 1993.

[Gal93] Jean Gallier. Constructive logics part i : A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110:249–339, 1993.

[Har87] Thérèse Hardin. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs.* Thèse de doctorat, Université de Paris VII, 1987.

[How92] Brian Howard. *Fixed Points and Extensionality in Typed Functional Programming Languages.* PhD thesis, Stanford University, August 1992.

[Hue76] Gérad Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse de doctorat d'état, Université Paris VII, 1976.

[Jay92] Colin Barry Jay. Long $\beta\eta$ normal forms and confluence (and its revised version). Technical Report ECS-LFCS-91-183, LFCS, University of Edimburgh, 1992.

[JG92] Colin Barry Jay and Neil Ghani. The virtues of eta-expansion. Technical Report ECS-LFCS-92-243, LFCS, University of Edimburgh, 1992.

[Jon87] Simon Peyton Jones. *The implementation of functional programming languages.* Prentice-Hall, 1987.

[Kes93] Delia Kesner. *La définition de fonctions par cas à l'aide de motifs dans des langages applicatifs.* Thèse de doctorat, Université de Paris XI, Orsay, december 1993.

[Min79] Grigori Mints. Teorija categorii i teoria dokazatelstv.I. *Aktualnye problemy logiki i metodologii nauky*, pages 252–278, 1979.

[Nes89] Dan Nesmith. An application of Klop's counterexample to a higher-order rewrite system. Seki-Report SR-94-06, Department of Computer Science, University of the Saarland, 1989.

[Pot81] Garrel Pottinger. The Church Rosser Theorem for the Typed lambda-calculus with Surjective Pairing. *Notre Dame Journal of Formal Logic*, 22(3):264–268, 1981.

[Pra71] Dag Prawitz. Ideas and results in proof theory. *Proceedings of the 2nd Scandinavian Logic Symposium*, pages 235–307, 1971.

[vO90] Vincent van Oostrom. Lambda calculus with patterns. Technical Report IR 228, Vrije Universiteit, Amsterdam, November 1990.

Appendix A: Evaluators in Natural Semantics Style

$$\frac{L \Downarrow_l \langle M, N\rangle \qquad M \Downarrow_l K}{\pi_1(L) \Downarrow_l K} \qquad \frac{L \Downarrow_l \langle M, N\rangle \qquad N \Downarrow_l K}{\pi_2(L) \Downarrow_l K}$$

$$\frac{}{\langle M, N\rangle \Downarrow_l \langle M, N\rangle} \qquad \frac{}{\lambda P : A.M \Downarrow_l \lambda P : A.M}$$

$$\frac{M[\mu x : A.M/x] \Downarrow_l K}{\mu x : A.M \Downarrow_l K}$$

$$\frac{M \Downarrow_l \lambda P : A.L \qquad \mathit{match}\ N\ \mathit{on}\ P \Downarrow_l \sigma \qquad L[\sigma] \Downarrow_l K}{M N \Downarrow_l K}$$

$$\frac{}{\mathit{match}\ L\ \mathit{on}\ x \Downarrow_l [L/x]} \qquad \frac{}{\mathit{match}\ L\ \mathit{on}\ \_ \Downarrow_l \emptyset}$$

$$\frac{L \Downarrow_l \langle M, N\rangle \quad \mathit{match}\ M\ \mathit{on}\ P \Downarrow_l \sigma \quad \mathit{match}\ N\ \mathit{on}\ Q \Downarrow_l \theta \quad \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \emptyset}{\mathit{match}\ L\ \mathit{on}\ \langle P, Q\rangle \Downarrow_l \sigma \cup \theta}$$

$$\frac{\mathit{match}\ L\ \mathit{on}\ P \Downarrow_l \sigma \quad \mathit{match}\ L\ \mathit{on}\ Q \Downarrow_l \theta \quad \mathit{Dom}(\sigma) \cap \mathit{Dom}(\theta) = \emptyset}{\mathit{match}\ L\ \mathit{on}\ P@Q \Downarrow_l \sigma \cup \theta}$$

Table 1: A Lazy Evaluator for the LWPP Calculus in Natural Semantics Style

$$\frac{L \Downarrow_e \langle M, N\rangle}{\pi_1(L) \Downarrow_e M} \qquad \frac{L \Downarrow_e \langle M, N\rangle}{\pi_2(L) \Downarrow_e N} \qquad \frac{M \Downarrow_e K \quad N \Downarrow_e L}{\langle M, N\rangle \Downarrow_e \langle K, L\rangle}$$

$$\frac{M[\mu x : A.M/x] \Downarrow_e K}{\mu x : A.M \Downarrow_e K} \qquad \frac{}{\lambda P : A.M \Downarrow_e \lambda P : A.M}$$

$$\frac{M \Downarrow_e \lambda P : A.J \qquad N \Downarrow_e K \qquad \mathit{Match}(K, P) = \sigma \qquad J[\sigma] \Downarrow_e L}{M N \Downarrow_e L}$$

$$\begin{array}{lll}
\mathit{Match}(N, _) & = \emptyset & \\
\mathit{Match}(N, x) & = [N/x] & \\
\mathit{Match}(N, P_1 @ P_2) & = \mathit{Match}(N, P_1) \cup \mathit{Match}(N, P_2) & \\
\mathit{Match}(\langle N_1, N_2\rangle, \langle P_1, P_2\rangle) & = \mathit{Match}(N_1, P_1) \cup \mathit{Match}(N_2, P_2) & \\
\mathit{Match}(M, P) & = \mathbf{fail} & \text{Otherwise}
\end{array}$$

Table 2: A Strict Evaluator for the LWPP Calculus in Natural Semantics Style

Appendix B

Lemma 4.1 *If $\Gamma \triangleright M : A$, then there is a term N such that $M \Longrightarrow^* N$ and $N :: A$.*

Proof. By definition of the structure of the type A.

$\square$

Theorem 2.15 (Redundancy of Layered and Wildcard Patterns) Any term M can be reduced to a term M' having no layered nor wildcard patterns.

Proof. Let $\stackrel{-}{\Longrightarrow}$ be the reduction $\Longrightarrow$ without recursion. We show that the $\stackrel{-}{\Longrightarrow}$-normal form of any term M contains no layered and no wildcard patterns. We proceed by induction on the structure of the term M.

- If $M = x$, the property is trivial.
- If $M = \langle M_1, M_2 \rangle$ or $M = \mu x : A.N$, the property holds by induction hypothesis.
- If $M = \lambda P : A.N$, then P is fully specified and so it has no layered and no wildcard patterns. As N is also in $\stackrel{-}{\Longrightarrow}$-normal form, the property holds by induction hypothesis.
- If $M = \pi_i(N)$, then N is only reducible at the root position, and thus all its proper subterms are in $\stackrel{-}{\Longrightarrow}$-normal form and do not contain layered nor wildcard patterns by induction hypothesis. Since N is of product type, it is not a λ-abstraction which concludes the proof.
- If $M = NL$, then the property holds for L by i.h. because it is a $\stackrel{-}{\Longrightarrow}$-normal form. By definition N is only reducible at the root position, and thus all its proper subterms are in $\stackrel{-}{\Longrightarrow}$-normal form and do not contain layered nor wildcard patterns by induction hypothesis. If N is not a λ-abstraction we are done, so suppose $N = \lambda P : A.N'$. Since L in in normal form it is fully specified by lemma 4.1 and so $Match(N, P)$ is defined and M is reducible. This leads to a contradiction so we can conclude that $N \neq \lambda P : A.N'$.

$\square$

Lemma 4.2 *The relation $\stackrel{*}{\Longleftrightarrow}$ is closed by context.*

Proof. In order to show that $\stackrel{*}{\Longleftrightarrow}$ is stable by context closure, it is sufficient to show the property for the relation $\Longleftrightarrow$. According to the definition 2.13 there are just two cases where this is not immediately evident but we show how to use $\Longleftrightarrow$, or more precisely $\Longleftarrow$, to obtain the result:

- If $M \stackrel{\eta}{\Longrightarrow} (\lambda P : A.MP)$, then the relation is immediately closed for all contexts *except* in the case where the terms are applied to a term L that is fully specified. By lemma 2.5 there is a substitution σ such that $Match(L, P) = \sigma$ and since $Var(P) \cap FV(M) = \emptyset$ holds by definition, then $M[\sigma] = M$ and:

$$(\lambda P : A.MP)L \stackrel{\beta}{\Longrightarrow} (MP)[\sigma] = (M[\sigma]P[\sigma]) = ML$$

- If $M \stackrel{sp}{\Longrightarrow} \langle \pi_1(M), \pi_2(M) \rangle$, then the relation is immediately closed for all the contexts *except* in the case where the terms are projected. We can use in this case the rules π_i as follows:

$$\pi_i(M) \stackrel{\pi_i}{\Longleftarrow} \pi_i(\langle \pi_1(M), \pi_2(M) \rangle)$$

□

Theorem 2.16 **($\Longrightarrow$ generates $\doteq$)** $M \doteq N$ if and only if $M \overset{*}{\Longleftrightarrow} N$.

Proof. If $M \overset{*}{\Longleftrightarrow} N$, then $M \Longleftrightarrow M_1 \Longleftrightarrow \ldots \Longleftrightarrow M_n \Longleftrightarrow N$, which implies $M \doteq M_1 \doteq \ldots \doteq M_n \doteq N$ because $\Longleftrightarrow$ is trivially included in $\doteq$. As $\doteq$ is an equivalence relation, then $M \doteq N$ holds by transitivity.

To prove that $M \doteq N$ implies $M \overset{*}{\Longleftrightarrow} N$, we proceed by induction on the depth of the derivation $M \doteq N$. If the depth of the derivation $M \doteq N$ is 0, then N is exactly M and $M \overset{*}{\Longleftrightarrow} M$ is trivial. Otherwise, we consider the following cases:

- If $M \doteq N$ comes from $N \doteq M$ by symmetry or comes from $M \doteq L$ and $L \doteq N$ by transitivity, then the results immediately follows by the induction hypothesis.
- If $M \doteq N$ comes from an equality $M' \doteq N'$ by a context closure, then by i.h. $M' \overset{*}{\Longleftrightarrow} N'$ and by lemma 4.2 we have $M \overset{*}{\Longleftrightarrow} N$.
- If $M \doteq N$ comes from an axiom, we have to show how to obtain $M \overset{*}{\Longleftrightarrow} N$ explicitly. We list only the problematic cases:
  - $\lambda P : A.M =_\eta \lambda Q : A.(\lambda P : A.M)Q$. The cases to consider are:
    - $P :: A$.
      - $Q :: A$. Then P and Q are necessarily α-equivalent, and if ψ is the renaming substitution between P and Q, then:

$$\lambda P : A.M =_\alpha \lambda Q : A.M[\psi] \overset{\beta}{\Longleftarrow} \lambda Q : A.(\lambda P : A.M)Q$$

      - Q is not fully specified by A (but $Q :\cdot A$). By lemma 2.5 $\mathit{Match}(P, Q) = \sigma$, so:

$$\begin{array}{lclcl} \lambda Q : A.(\lambda P : A.M)Q & \overset{\eta}{\Longrightarrow} & \lambda P : A.(\lambda Q : A.(\lambda P : A.M)Q)P & \overset{\beta}{\Longrightarrow} & \\ \lambda P : A.(\lambda P : A.M)Q[\sigma] & = & \lambda P : A.(\lambda P : A.M)P & \overset{\beta}{\Longrightarrow} & \lambda P : A.M \end{array}$$

    - P is not fully specified by A.
      - $Q :: A$. Then

$$\lambda P : A.M \overset{\eta}{\Longrightarrow} \lambda Q : A.(\lambda P : A.M)Q$$

      - Q is not fully specified by A. Let us take $R :: A$ and let σ be the substitution such that $\mathit{Match}(R, Q) = \sigma$ (it exists by lemma 2.5). We then have :

$$\begin{array}{ccc} \lambda Q : A.(\lambda P : A.M)Q & & \\ \Downarrow_\eta & & \\ \lambda R : A.(\lambda Q : A.(\lambda P : A.M)Q)R & & \lambda P : A.M \\ \Downarrow_\beta & & \Downarrow_\eta \\ \lambda R : A.(\lambda P : A.M)Q[\sigma] & = & \lambda R : A.(\lambda P : A.M)R \end{array}$$

  - $\langle M, N\rangle =_{sp} \langle \pi_1(\langle M, N\rangle), \pi_2(\langle M, N\rangle)\rangle$. We cannot apply an expansion to a term that is a pair, but we can use the reductions π_i as follows:

$$\langle M, N\rangle \overset{\pi_2}{\Longleftarrow} \langle M, \pi_2(\langle M, N\rangle)\rangle \overset{\pi_1}{\Longleftarrow} \langle \pi_1(\langle M, N\rangle), \pi_2(\langle M, N\rangle)\rangle$$

□