The Journal of Functional and Logic Programming

The MIT Press

Volume 1997, Article 4

13 June, 1997

ISSN 1080–5230. MIT Press Journals, 55 Hayward St., Cambridge, MA 02142, USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://www.cs.tu-berlin.de/journal/jflp/*
- *http://mitpress.mit.edu/jrnls-catalog/jflp.html*
- *gopher.mit.edu*
- anonymous *ftp* at *mitpress.mit.edu*

The Journal of Functional and Logic Programming is a peer-reviewed and electronically published scholarly journal that covers a broad scope of topics from functional and logic programming. In particular, it focuses on the integration of the functional and the logic paradigms as well as their common foundations.

Reasoning about Redundant Patterns

Delia Kesner

13 June, 1997

Abstract

The extensional version of the simply typed λ-calculus with product types enriched with *layered*, *wildcard*, and *product patterns* is studied. Extensionality is expressed by the surjective pairing axiom and a generalization of the η-conversion to patterns. Two different *confluent* reduction systems, called $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ respectively, are obtained by turning the extensional axioms as *expansion* rules, and then adding some restrictions to these expansions to avoid reduction loops. It is shown that only layered and wildcard patterns are redundant in $\xrightarrow{lw}$, while product patterns are unnecessary in $\xrightarrow{lwp}$. Confluence of both reduction systems is proven by the composition of modular properties of the systems' extensional and nonextensional parts. Recursion is also added to both systems by keeping the modularity of the confluence property.

1 Introduction

Pattern-matching in function definitions is one of the most popular features of functional languages, as it allows for the specification of function behavior by case, according to the argument form. The left-hand sides of function definitions are usually expressed using *layered*, *wildcard*, and *product patterns* (LWPP), as for example in the Caml Light program [L+93] displayed in Figure 1, where the function `cons_new_pair` takes a pair (m_1,m_2) of type `'a * 'b` and yields the pair $(m_1,(m_1,m_2))$.

The function `cons_new_pair` can also be written in the calculus studied in this paper, called the LWPP calculus, which is just the simply typed λ-calculus with product types that are enriched with layered, wildcard, and

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

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

# cons_new_pair (2,true);;

- :  int * (int * bool) = 2, (2, true)
```

Figure 1: Caml Light example.

product patterns. Using these pattern constructors, abstractions of functions can be made not only with respect to variables, but also with respect to patterns, thereby specifying the form of a function's arguments. Other formalisms for λ-calculi with nested patterns can be found in [vO90, PJ87, How92, KPT96]. In particular, the LWPP calculus can also be viewed as a subcalculus of the Typed Pattern Calculus (TPC) that was proposed in [BTKP93, KPT96]. 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 proof system for intuitionistic propositional logic. For this reason, 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.

There is 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 function `cons_new_pair` in the LWPP calculus.

Example 1

$$\lambda\langle x, y\rangle : A \times B.\langle x, \langle x, y\rangle\rangle \tag{1}$$

$$\lambda x : A \times B.\langle \pi_1(x), x\rangle \tag{2}$$

$$\lambda x@\langle y, \_\rangle : A \times B.\langle y, x\rangle \tag{3}$$

End of Example 1 □

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 must be computationally reasonable, and the reduction system must be confluent. In this way, the reduction rules simultaneously play two roles: 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:

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

The existence of patterns in the scope of λ-abstractions suggests a natural extension of the η-axiom to express equalities between pattern abstractions [CCMS86]. The new axiom can be specified as follows:

$$\lambda P.(MP) = M \qquad \begin{array}{l}\text{provided } P \text{ contains only variables and pairs,}\\ \text{and the variables of } P \text{ are not free in } M\end{array} \qquad (\eta_p)$$

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. It can be specified by the following equality:

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

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

When one wants to turn these equalities into rules, there are two different choices. When turning them from left to right as *contractions*, simply state that a term $\langle \pi_1(M), \pi_2(M)\rangle$ is more complex than M, and a term $\lambda P.MP$ is more complex than M. Otherwise, the equalities can be turned into *expansions*, whereby 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 must really be 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 a contraction causes confluence to fail [Nes89]. Moreover, the contractive interpretation of the η_p-axiom for patterns breaks confluence, as the following example shows.

Example 2

$$\begin{array}{c} \lambda\langle x,y\rangle : A\times B.\langle x,y\rangle \\ \uparrow_{\beta} \\ \lambda\langle x,y\rangle : A\times B.(\lambda z : A\times B.z)\langle x,y\rangle \\ \downarrow_{\eta_p} \\ \lambda z : A\times B.z \end{array}$$

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.

End of Example 2 □

Fortunately, this inconvenient situation can be overcome by turning the extensional equalities into *expansion* rules. This idea was suggested by Prawitz [Pra71] in the field of proof theory, and then further studied by Huet [Hue76] in the field of higher-order unification, and by Mints [Min79], who was motivated by problems in category theory. Expansions, however, seem to have passed unnoticed for a long time, until recent years, when it has been shown that they can be perfectly combined with typed λ-calculus [CK94, Aka93, Dou93, Jay92, JG95, Čub92], and with many other reduction rules such as recursion [CK94, Dou93], sums [CK94, Dou93], algebraic rewriting systems [CK96], and second-order λ-calculus [CK95].

In this paper, the expansive interpretation of extensional axioms in a typed λ-calculus with patterns is investigated. Mainly inspired by [Aka93, CK94], we use expansion rules to provide two different confluent reduction systems, called *lwp* and *lw*, respectively, for the extensional typed-pattern calculus with layered, wildcard, and product patterns. One of these reduction systems, *lw*, has already been investigated in [Kes94] and allows the characterization of the so-called η-long normal forms used to perform higher unification in a pattern framework [FL96]. The other system, *lwp*, corresponds to the classical extensional reduction system for typed λ-calculus, with the difference that objects now belong to a complex calculus with patterns.

It is shown that the *lw* system allows for the rewriting of any function $\lambda P : A.M$ into another equivalent function $\lambda P' : A.M'$ such that the pattern P' contains only variables and pairs, while the system *lwp* allows for the rewriting of any function into another equivalent one without patterns. This corresponds to the fact that wildcard and layered patterns are always redundant, while product patterns are only redundant in *lwp*. At the same time, these results yield another interpretation: we can discard at compile time such redundant patterns (using our reduction systems) as a form of program optimization that will then be evaluated by simpler computation rules.

Also it is shown that the reduction systems *lw* and *lwp* are adequate for the axiomatic theory, i.e., equalities are not lost by the notion of restricted reduction relations that is defined in the paper. In particular, the three functions of Example 1 can be proven equal, since they all have the same normal form.

The confluence of both *lw* and *lwp* is shown by a divide-and-conquer lemma that requires different properties: the confluence of the system without expansions (the relation noted $\xrightarrow{B}$), the confluence and strong normalization of the system having only expansions (noted $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$, respectively), and the property of *projection of B with respect to E and $\widehat{E}$ normal forms*—every time a term M reduces to a term N in the system without expansions, there is a nonempty $\xrightarrow{B}$ reduction sequence from the E (resp. $\widehat{E}$) normal form of M to the E (resp. $\widehat{E}$) normal form of N. This technique clarifies the relation between the reduction $\xrightarrow{B}$ and the normalization with respect to the expansion rules, and it is independently interesting because the explicit descriptions of E and $\widehat{E}$ normal forms make it easier to understand the behavior of expansions in a pattern framework. Besides that, a clean and formal proof of strong normalization of expansive η and *sp* is provided, which has always been given in the literature by rather informal arguments.

The calculus is introduced in Section 2, including the typing rules (Section 2.1), the equalities (Section 2.2), and the reduction systems (Section 2.3). Section 3 shows the adequacy of the reduction systems with respect to the equational axiomatization. Proofs are provided to show that wildcard and layered patterns can be eliminated in *lw*, whereas product patterns can also be eliminated in *lwp*. The main properties of the reduction relations are studied in Section 4: the confluence of $\xrightarrow{B}$ on the set of all the terms and the strong normalization of $\xrightarrow{B}$ on the set of typed terms (Section 4.1), the strong normalization and confluence of $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$ (Section 4.2), and projection of B with respect to E and $\widehat{E}$ normal forms (Section 4.3). Finally, Section 5 shows how to add recursion to both calculi by keeping the confluence property. The paper concludes with ideas for further work.

2 The LWPP Calculus

2.1 Terms and Types

It is time to introduce the LWPP calculus. The set of *types* of this calculus is the least set containing a denumerable set of base types (as, for example, natural numbers and Booleans), and closed with respect to formation of function and product types, i.e., if A and B are types, then $A \to B$ and $A \times B$ are also types.

To introduce the syntax of patterns and terms, a denumerable set of variables $\{x, y, z, \ldots\}$ is first fixed. The well-typed terms are given by the set of *typing judgments* in Figure 2 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 judgment $P_1 : A_1 \ldots P_n : A_n \triangleright M : A$ 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. In general, M is of type A if there is a pattern-type assignment Γ such that $\Gamma \triangleright M : A$ is either an axiom or follows from axioms by the application of the typing rules. The term M is said to be *well typed* if there exists a type A such that M is of type A. In general, the *pattern P is of type A* if there exist Γ, M, and B such that $\Gamma, P : A \triangleright M : B$.

The left rules $\times l$, *layered*, and *wildcard* are used to introduce patterns (called *complex* patterns) that correspond to devices long used in programming languages, and the right rules $\times r$, *app*, *abs*, and *proj$_i$* introduce the terms of the typed λ-calculus. In particular, notice that the *abs* rule allows for the abstraction of a function not only with respect to a variable, but also with respect to a pattern. The layered pattern $\_@\_$ introduced by the *layered* rule is used to uniquely name two different patterns of the same type; 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 $\times l$ rule is the usual product pattern (also called the pair pattern).

The notion of *context* is used for terms with "holes," assuming that holes do not appear in the patterns of λ-abstractions. For example, $C[]$ denotes a

Axiom:

$$\frac{\text{the } x_j\text{'s are pairwise distinct}}{x_1 : A_1, \ldots, x_n : A_n \triangleright x_i : A_i}\ (axiom)$$

Left rules:

$$\frac{\Gamma \triangleright M : B}{\Gamma, \_ : A \triangleright M : B}\ (wildcard) \qquad \frac{\Gamma, P_1 : A, P_2 : A \triangleright M : B}{\Gamma, P_1 @ P_2 : A \triangleright M : B}\ (layered)$$

$$\frac{\Gamma, P_1 : A_1, P_2 : A_2 \triangleright M : B}{\Gamma, \langle P_1, P_2 \rangle : A_1 \times A_2 \triangleright M : B}\ (\times l)$$

Right rules:

$$\frac{\Gamma \triangleright M_1 : A_1 \quad \Gamma \triangleright M_2 : A_2}{\Gamma \triangleright \langle M_1, M_2 \rangle : A_1 \times A_2}\ (\times r) \qquad \frac{\Gamma \triangleright M : A \to B \quad \Gamma \triangleright N : A}{\Gamma \triangleright (MN) : B}\ (app)$$

$$\frac{\Gamma, P : A \triangleright M : B}{\Gamma \triangleright \lambda P : A.M : A \to B}\ (abs)$$

$$\frac{\Gamma \triangleright M : A \times B}{\Gamma \triangleright \pi_1(M) : A}\ (proj_1) \qquad \frac{\Gamma \triangleright M : A \times B}{\Gamma \triangleright \pi_2(M) : B}\ (proj_2)$$

Figure 2: Typing rules for the LWPP calculus.

context, and $C[M]$ represents the context $C[]$ where all the holes are replaced by M. Thus, $C[M]$ denotes a term. The term a is defined to be *applied* in any term having the form $C[(ab)]$, and *projected* in any term having the form $C[\pi_i(a)]$. Types of patterns in λ-abstractions may be omitted when they are clear from the context, as when 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 constructed by the rules in Figure 2 are linear). $Var(P)$ is the set of variables that occurs in the pattern P; $FV(M)$ the set of free variables that occurs in the term M; and $BV(M)$ is the set of bound variables in the term M. The phrase $[N_1, \ldots, N_n/x_1, \ldots, x_n]$ (often abbreviated $[\overline{N}/\overline{x}]$) indicates 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, $\mathrm{Dom}(\theta) \cap \bigcup_{x \in \mathrm{Dom}(\theta)} FV(\theta(x)) = \emptyset$ is assumed.

As opposed to the simply typed λ-calculus, here a judgment may have several derivations. As an example, two different derivations for the judgment $\langle x, y\rangle : A \times B \rhd \langle x, y\rangle : A \times B$ are shown:

$$\dfrac{\dfrac{x : A, y : B \rhd x : A}{\langle x, y\rangle : A \times B \rhd x : A}\,(\times l) \qquad \dfrac{x : A, y : B \rhd y : B}{\langle x, y\rangle : A \times B \rhd y : B}\,(\times l)}{\langle x, y\rangle : A \times B \rhd \langle x, y\rangle : A \times B}\,(\times r)$$

$$\dfrac{\dfrac{x : A, y : B \rhd x : A \quad x : A, y : B \rhd y : B}{x : A, y : B \rhd \langle x, y\rangle : A \times B}\,(\times r)}{\langle x, y\rangle : A \times B \rhd \langle x, y\rangle : A \times B}\,(\times l)$$

However, this is not as problematic as the left rules *layered*, *wildcard*, and $\times l$ commute with all the other rules. This property, called the *commutation property*, is illustrated by the following example (all the other cases are left to the reader). The $\times l$ rule commutes with the *app* rule as the derivation

$$\dfrac{\dfrac{\Gamma, P : A, Q : B \rhd M : C \to D \quad \Gamma, P : A, Q : B \rhd N : C}{\Gamma, P : A, Q : B \rhd (MN) : D}\,(app)}{\Gamma, \langle P, Q\rangle : A \times B \rhd (MN) : D}\,(\times l)$$

can be transformed into

$$\dfrac{\dfrac{\Gamma, P : A, Q : B \rhd M : C \to D}{\Gamma, \langle P, Q\rangle : A \times B \rhd M : C \to D}\,(\times l) \quad \dfrac{\Gamma, P : A, Q : B \rhd N : C}{\Gamma, \langle P, Q\rangle : A \times B \rhd N : C}\,(\times l)}{\Gamma, \langle P, Q\rangle : A \times B \rhd (MN) : D}\,(app)$$

As a consequence, it is clear that for every *nonvariable* term M, there exists *exactly one* rule such that every judgment $\Gamma \rhd M : E$ has a derivation ending with the application of that rule. Indeed, for any judgment of a pair $\langle M, N\rangle$, there is a derivation ending with the $\times l$ rule; for any judgment 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*), and finally, the $proj_i$ rules correspond to the terms of the form $\pi_i(M)$.

Only patterns built from variables and the pair constructor are used to define the extensional axioms. It is interesting that those axioms are sufficient to express equalities between terms having layered and wildcard patterns, such as those of Example 1. Before the definition of the equality axioms, the notion of *specification* and *full specification* of a pattern P by a type A, writing $P \gg A$ and $P :: A$, respectively, is introduced. The notion of specification avoids the association of nonproduct types to pair patterns; beyond this, the notion of full specification forbids 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 1 (Specification of Patterns) *For every variable x and type A, we have $x \gg A$, and whenever $P_1 \gg A_1$ and $P_2 \gg A_2$, then $\langle P_1, P_2\rangle \gg A_1 \times A_2$.*

Notice that patterns such as _ and $P_1@P_2$ are never specified by a type, and that a pattern P that has only variables and pairs is specified by a type A if and only if it is of type A.

Definition 2 (Full Specification of Patterns) *For every variable x and nonproduct 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 3 (Full Specification of Terms) *A term M is* fully specified (FS) *if M is fully specified by its type, and M is* fully specified *by a type A (written $M :: A$) if and only if M is of type A and one of the following cases holds:*

- $M \neq \langle M_1, M_2\rangle$ *and* $A \neq A_1 \times A_2$*, or*
- $M = \langle M_1, M_2\rangle$*,* $A = A_1 \times A_2$*,* $M_1 :: A_1$ *and* $M_2 :: A_2$*.*

Example 3 Let A and B be two base types, z and $\langle x, y\rangle$ be two patterns, and $\pi_1(w)$ and $\langle \pi_1(w), z\rangle$ two terms. Then $z \gtrdot A \times B$ and $\langle x, y\rangle \gtrdot A \times B$, but z is not fully specified by $A \times B$, nor is $\langle x, y\rangle$ specified by 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 is not fully specified by $A \times B$, nor is $\langle w, z\rangle$ fully specified by A.

End of Example 3 $\square$

A fully specified pattern is also a specified one, and the full-specification property is stable by substitution and reduction. Notice also that for every pattern P specified by A, $P : A \rhd P : A$.

Finally, the following lemma shows that patterns can be added or removed from pattern type assignments (PTA).

Lemma 1 (Removing Patterns from PTA) *If* $\Gamma, P : A \rhd M : C$*, and* $Var(P) \cap FV(M) = \emptyset$*, then* $\Gamma \rhd M : C$*.*

Proof of Lemma 1 (Removing Patterns from PTA) By induction on the height of the derivation $\Gamma, P : A \rhd M : C$.

Proof of Lemma 1 $\square$

Lemma 2 (Adding Patterns to PTA) *If* $Var(P) \cap Var(\Gamma) = \emptyset$*,* $\Gamma \rhd N : A$*, and* P *is a linear pattern of type* B*, then* $\Gamma, P : B \rhd N : A$*.*

Proof of Lemma 2 (Adding Patterns to PTA) By induction on the structure of P.

Proof of Lemma 2 $\square$

Below, standard notions from the theory of rewriting systems [Klo80, DJ90] will be used, such as redex, normal form, confluence, weak confluence, and strong normalization.

Notation 1 *The transitive and the reflexive transitive closure of any reduction relation* $\xrightarrow{R}$ *are noted* $\xrightarrow{R}^+$ *and* $\xrightarrow{R}^*$*, respectively, while* M^S *denotes the* S*-normal form of* M*.*

2.2 Equalities

A function Match(M, P) is defined on every term M and every pattern P; this permits the generalization of the usual β-rule to the case of patterns. This function clearly explains the operational semantics of layered, wildcard, and product patterns by the notion of matching substitution, which is defined as follows:

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

Match$(_, _)$ is well defined in terms of the operation $\cup$, because all patterns are linear according to the typing rules in Figure 2. Using this definition reveals the existence of matching substitutions between fully specified terms and specified patterns, and also that the substitution lemma holds for our calculus.

Lemma 3 *If P is a pattern of type A, and N is a term fully specified by A, then there is a substitution σ such that $Match(N, P) = \sigma$. Moreover, if P is specified by A, then $\sigma(P) = N$.*

Proof of Lemma 3 By induction on the structure of P.

- If P is a variable or $\_$, the property is trivial.
- If P is a layered pattern $P_1@P_2$, then by induction hypothesis there are substitutions σ_1 and σ_2 such that Match$(N, P_i) = \sigma_i$, for $i = 1, 2$. Since P is linear, then $\sigma = \sigma_1 \cup \sigma_2$ satisfies the property.
- If P is a product pattern $\langle P_1, P_2\rangle$, then A is necessarily a product type $A_1 \times A_2$, and so N is a pair $\langle N_1, N_2\rangle$. By induction hypothesis there are substitutions σ_1 and σ_2 such that Match$(N_i, P_i) = \sigma_i$, for $i = 1, 2$. Since P is linear, then $\sigma = \sigma_1 \cup \sigma_2$, and so Match$(N, P) = \sigma$. If P is specified by A, then $\sigma(P) = \sigma(\langle P_1, P_2\rangle) =_{\text{i.h.}} \langle \sigma(P_1), \sigma(P_2)\rangle = \langle N_1, N_2\rangle = N$.

Proof of Lemma 3 $\Box$

Lemma 4 (Substitution Lemma) *If* $\Gamma, P : C \triangleright M : A$, $\Gamma \triangleright N : C$, *and* $Match(N, P) = \theta$, *then* $\Gamma \triangleright M[\theta] : A$.

Proof of Lemma 4 (Substitution Lemma) The proof is by induction on the height of the derivation $\Gamma, P : B, \Gamma \triangleright M : A$:

- If $\Gamma, P : B \triangleright M : A$ is an axiom $\Gamma, y : C \triangleright x : A$, there are two cases:
  - If $y \equiv x$, then $\theta = [N/x]$, $x[\theta] = N$, and the property holds by hypothesis.
  - If $y \not\equiv x$, then $x[N/y] = x$, and the property holds since $\Gamma \triangleright x : A$ is also an axiom.
- If $\Gamma, P : C \triangleright M : A$ is not an axiom, then the property holds by the induction hypothesis.

Proof of Lemma 4 $\square$

Equalities between terms of the calculus may now be determined. The α-conversion is the most natural one, because it allows the renaming of bound variables. If ψ is a renaming substitution, the α-equality for λ-abstractions corresponds to the equation:

$$\lambda P : A.M =_\alpha \lambda P[\psi] : A.M[\psi], \text{ if } Var(P[\psi]) \cap (FV(M) - Var(P)) = \emptyset \text{ and } Var(P[\psi]) \cap BV(M) = \emptyset \qquad (\alpha)$$

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, the calculus is equipped with the following equalities between terms:

$$(\lambda P : A.M)N =_\beta M[\theta] \qquad \text{if Match}(N, P) = \theta \qquad (\beta)$$

$$\pi_i\langle M_1, M_2\rangle =_{\pi_i} M_i \qquad \text{for } i = 1, 2 \qquad (\pi_i)$$

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

Definition 4 (LWPP Calculus) *The* LWPP calculus *is equipped with the typing rules in Figure 2 and the conversion* $=_{LWPP}$.

The following conversion axioms are the extensional rules of the LWPP calculus, where the axiom η_p is the natural extension to patterns of the η-equality of the simply typed λ-calculus.

$$M =_{\eta_p} \lambda P : A.(MP) \qquad \text{if } \mathit{Var}(P) \cap FV(M) = \emptyset, \; M \text{ is of type } A \to B, \text{ and } P \text{ is specified by } A \qquad (\eta_p)$$

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

Notice that the η_p-axiom generalizes the usual η-axiom, as P may also be an arbitrary variable that is not free in M. The η_p-equality also implies the categorical uniqueness of the product type: indeed, taking $fst \equiv \lambda\langle x_1, x_2\rangle.x_1$ and $snd \equiv \lambda\langle x_1, x_2\rangle.x_2$, then it can be shown that

$$\begin{array}{l}
\lambda x : A \times B.\langle \mathit{fst}\ x, \mathit{snd}\ x\rangle =_{\eta_p} \\
\lambda\langle u, v\rangle : A \times B.(\lambda x : A \times B.\langle \mathit{fst}\ x, \mathit{snd}\ x\rangle)\langle u, v\rangle =_\beta \\
\lambda\langle u, v\rangle : A \times B.\langle \mathit{fst}\ \langle u, v\rangle, \mathit{snd}\ \langle u, v\rangle\rangle =_\beta \\
\lambda\langle u, v\rangle : A \times B.\langle u, \mathit{snd}\ \langle u, v\rangle\rangle =_\beta \\
\lambda\langle u, v\rangle : A \times B.\langle u, v\rangle =_\beta \\
\lambda\langle u, v\rangle : A \times B.(\lambda x : A \times B.x)\langle u, v\rangle =_{\eta_p} \\
\lambda x : A \times B.x
\end{array}$$

to conclude that

$$z =_\beta (\lambda x : A \times B.x)z =_{\beta,\eta_p} (\lambda x : A \times B.\langle \mathit{fst}\ x, \mathit{snd}\ x\rangle)z =_\beta \langle \mathit{fst}\ z, \mathit{snd}\ z\rangle$$

Definition 5 (Extensional LWPP Calculus) *The* extensional LWPP calculus *contains the typing rules in Figure 2 and the conversion* $\doteq$*, which is defined as the* $=_{LWPP}$ *equality enriched with* $=_{\eta_p}$ *and* $=_{sp}$*.*

It is interesting to remark that extensional rules are not always of interest. Take for example a nonterminating term M; then the equality $M =_{sp} \langle \pi_1(M), \pi_2(M)\rangle$ does not hold in general since $\langle \bot, \bot\rangle = \bot$ is not true in a nonstrict model.

2.3 The Relations $\xrightarrow{lw}$ and $\xrightarrow{lwp}$

While equations correspond to the axiomatic semantics of the calculus, reduction rules give the operational semantics as a form of symbolic computation, i.e., they model program execution. For that, a reduction system must 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: therefore confluence must be insured by means of an appropriate reduction system.

Traditionally, a reduction system is obtained from the equational system by simply orienting the equations in a convenient way. But this is not the case for this calculus, as the following will explain; a more delicate procedure is required.

To obtain a reduction system for the LWPP calculus, the *nonextensional rules* are first turned from left to right in the following way.

Definition 6 (The Nonextensional Rules)

$$(\lambda P : A.M)N \longrightarrow M[\theta] \qquad \textit{if } Match(N, P) = \theta \qquad (\beta)$$
$$\pi_i(\langle M_1, M_2\rangle) \longrightarrow M_i \qquad \textit{for } i = 1, 2 \qquad (\pi_i)$$

When turning the *extensional equalities* into reduction rules, the contractive choice breaks confluence (Example 2) while the expansive choice breaks strong normalization (Example 4).

Example 4

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

End of Example 4 □

Even if a fixpoint operator is added (in Section 5) that gives a natural mechanism for producing nonterminating sequences, it is preferable to keep expansions as strongly normalizing rules, to provide a characterization of (η, sp)-normal forms that are needed when working with η-long normal forms in higher-order unification [Hue76, FL96]. Strong normalization of the extensional rules is recovered when some restrictions are imposed upon them. To define an appropriate reduction system for this calculus, the loops generated by the arbitrary application of an extensional rule turn as an expansion is analyzed. There are two different situations in which such loops may arise, called *structural loops* and *contextual loops*. 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*.

2.3.1 Loops Caused by the Rule sp

The structural loop caused by sp-expansions appears in Example 4, and can be avoided by forbidding expansions on terms that are already pairs:

$$M \xrightarrow{sp} \langle \pi_1(M), \pi_2(M) \rangle \quad \text{if } M \text{ is of type } A \times B \text{ and } M \text{ is not a pair} \qquad (sp)$$

The contextual loops for sp are produced by the application of an sp-expansion to a subterm that is projected, i.e., to a subterm that is the argument of a projection:

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

To avoid these loops, the sp-expansions will not be allowed on this kind of terms, yielding the following reduction relation.

Definition 7 (The Relation $\xrightarrow{sp}$) *The sp-expansion is the reflexive and transitive closure of the following reduction step:*

$$C[M] \xrightarrow{sp} C[\langle \pi_1(M), \pi_2(M) \rangle] \quad \textit{if } M \textit{ is of type } A \times B, \; M \textit{ is not a pair, and } M \textit{ is not projected in } C[M] \qquad (sp)$$

2.3.2 Loops Caused by Rule η_p

The structural loops caused by the application of the η_p-expansion will be investigated here, starting with a case that is also problematic in the typed λ-calculus:

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

This situation can be reproduced in the presence of more complicated patterns, where the loop starts in a term having a pair as abstraction.

$$\lambda\langle x, y\rangle : A \times B.M \rightsquigarrow \lambda\langle w, z\rangle : A \times B.(\lambda\langle x, y\rangle : A \times B.M)\langle w, z\rangle \rightsquigarrow \lambda\langle w, z\rangle : A \times B.M[w/x][z/y] =_\alpha \lambda\langle x, y\rangle : A \times B.M$$

In general, each time M is a λ-abstraction, $\lambda Q : A.N$, and it reduces via η_p to a term $\lambda P : A.(\lambda Q : A.N)P$ such that the pattern P is "as specific as" Q (i.e., α-equivalent to Q), a β-step to close the loop can always be performed. As a consequence, whenever an expansion is applied to a λ-abstraction $\lambda Q : A.N$, it must introduce a pattern P that is either strictly "less specific" or "more specific" than Q.

However, this restriction is insufficient, as the following less-trivial example shows, where the first expansion rule introduces a more-specific pattern $\langle x, y\rangle$ while the second expansion introduces a less-specific pattern w:

$$\begin{array}{ll}
\lambda z : A \times B.\langle \pi_1(z), \pi_2(z)\rangle & \leadsto \\
\lambda\langle x, y\rangle : A \times B.(\lambda z : A \times B.\langle \pi_1(z), \pi_2(z)\rangle)\langle x, y\rangle & \leadsto \\
\lambda\langle x, y\rangle : A \times B.\langle \pi_1(\langle x, y\rangle), \pi_2(\langle x, y\rangle)\rangle & \leadsto^* \\
\lambda\langle x, y\rangle : A \times B.\langle x, y\rangle & \leadsto \\
\lambda w : A \times B.(\lambda\langle x, y\rangle : A \times B.\langle x, y\rangle)w & \leadsto \\
\lambda w : A \times B.(\lambda\langle x, y\rangle : A \times B.\langle x, y\rangle)\langle \pi_1(w), \pi_2(w)\rangle & \leadsto \\
\lambda w : A \times B.\langle \pi_1(w), \pi_2(w)\rangle & =_\alpha \\
\lambda z : A \times B.\langle \pi_1(z), \pi_2(z)\rangle &
\end{array}$$

This suggests that it is necessary to chose a canonical form of applying expansions to λ-abstractions: one always uses either the less-specific patterns (that is, variables) or the more-specific ones. The first possibility gives the following reduction rule:

$$M \leadsto \lambda x : A.(Mx) \quad \begin{array}{l} \text{if } x \notin FV(M), \\ M \text{ is of type } A \to B, \text{ and} \\ M \neq \lambda y : A.N \end{array} \qquad (\eta_1)$$

The second possibility attempts to capture as much information as possible using the notion of full specification that now comes into play:

$$M \leadsto \lambda P : A.(MP) \quad \begin{array}{l} \text{if } Var(P) \cap FV(M) = \emptyset, \\ P \text{ is fully specified by } A, \\ M \text{ is of type } A \to B, \text{ and} \\ M = \lambda Q : A.N \text{ implies } Q \not\colon A \end{array} \qquad (\eta_2)$$

In [Kes94], the approach corresponding to the η_2-rule is taken. In this paper, the other possible interpretation corresponding to the η_1-rule is explored, exactly as in the case of the classical η-expansion defined for the typed

λ-calculus. Here it is shown that both possible operational interpretations of η_p are well behaved with respect to the confluence and strong normalization properties, and some particular properties of redundancy of patterns in each system are studied.

2.3.3 The Reduction System *lwp*

The last restriction in the η_1-rule introduced above is sufficient to avoid structural loops of η_p, but it is not sufficient to avoid *contextual* loops because of the application of an η_p-expansion to a term M that is applied to another term N: indeed, $MN \leadsto (\lambda x.Mx)N \leadsto MN$.

To break this kind of loop, an η_1-expansion step on a subterm that is applied to another term is forbidden. The resulting rule is simply called the η-rule, as is the same rule used in the simply typed λ-calculus.

Definition 8 (Relation $\xrightarrow{\eta}$) *The η-expansion rule is the reflexive and transitive closure of the following reduction step:*

$$C[M] \xrightarrow{\eta} C[\lambda x : A.(Mx)] \quad \begin{array}{l} \text{if } x \notin FV(M), \\ M \text{ is of type } A \to B, \\ M \neq \lambda y : A.N, \text{ and} \\ M \text{ is not applied in } C[M] \end{array} \qquad (\eta)$$

It can be immediately verified that all of the previously shown loops are now ruled out.

Definition 9 (Relation $\xrightarrow{lwp}$) *The one-step relation $\xrightarrow{lwp}$ is the union of the reduction rules β, π_1, π_2, η, and sp. Notice that $\xrightarrow{lwp}$ is not a congruence, since $M \xrightarrow{lwp} N$ does not imply $ML \xrightarrow{lwp} NL$ because of the η-rule, and $M \xrightarrow{lwp} N$ does not imply $\pi_i(M) \xrightarrow{lwp} \pi_i(N)$ because of the sp-rule.*

2.3.4 Reduction System *lw*

The last restriction in the η_2-rule introduced above is also sufficient to avoid the so-called structural loops of η_p, but it is not sufficient to avoid the contextual loops as the following example shows:

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

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 η_2-rule to the term M produces (by Lemma 3) a matching substitution θ such that $\theta(P) = L$. The loop starting at ML is of the form:

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

To break this kind of loop, an η_2-expansion is forbidden on a subterm that is applied to a *fully specified* term.

Notice that when L is not fully specified, the η_2-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, $MN \rightsquigarrow (\lambda\langle z, w\rangle : A \times B.M\langle z, w\rangle)N$.

The restrictions are now satisfied by the following reduction rule.

Definition 10 (Relation $\xrightarrow{\widehat{\eta}}$) *The $\widehat{\eta}$-expansion rule is the reflexive and transitive closure of the following reduction step:*

$$C[M] \xrightarrow{\widehat{\eta}} C[\lambda P : A.(MP)] \quad \begin{array}{l} \textit{if } Var(P) \cap FV(M) = \emptyset, \\ P \textit{ is fully specified by } A, \\ M \textit{ is of type } A \to B, \\ M = \lambda Q : A.N \textit{ implies } Q \not\triangleleft{:}A, \textit{ and} \\ M \textit{ is not applied in } C[M] \textit{ to an FS term} \end{array} \qquad (\widehat{\eta})$$

It can also be verified that all of the previously shown loops are now ruled out.

Definition 11 (Relation $\xrightarrow{lw}$) *The one-step reduction relation $\xrightarrow{lw}$ is the union of the reduction rules β, π_1, π_2, $\widehat{\eta}$, and sp. The relation $\xrightarrow{lw}$ is not a congruence, due to the same reasons given in Definition 9.*

3 Adequacy of Expansions and Redundancy of Patterns

In this section the adequacy of the extensional rules is shown by proving that the equality $\doteq$ can be generated by the reflexive, symmetric, and transitive closure of $\xrightarrow{C}$, for $C \in \{lwp, lw\}$. The second result of this section is related to redundancy of patterns: indeed, in the reduction system $\xrightarrow{lw}$, layered and wildcard patterns turn out to be unnecessary, and product patterns may be omitted in the system $\xrightarrow{lwp}$. Following are some properties that are enjoyed by the reduction relations.

Using Lemma 4 and the commutation property, it is easy to show that the one-step reduction relations $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ preserve types.

Theorem 1 (Subject Reduction) *If $\Gamma \triangleright M : A$ and $M \xrightarrow{C} M'$ where C is lwp or lw, then $\Gamma \triangleright M' : A$.*

Proof of Theorem 1 (Subject Reduction) If $M \xrightarrow{C} M'$ is a root reduction step, then the proof proceeds by cases, otherwise one can reason by induction on the structure of M. Following are the cases where $M \xrightarrow{C} M'$ is a root expansion step:

- $M \xrightarrow{sp} \langle \pi_1(M), \pi_2(M) \rangle$. In this case $A \equiv A_1 \times A_2$, for $i = 1, 2$, $\Gamma \triangleright \pi_i(M) : A_i$, and $\Gamma \triangleright \langle \pi_1(M), \pi_2(M) \rangle : A_1 \times A_2$ hold by definition.
- $M \xrightarrow{\eta} \lambda x : B.Mx$. In this case $A \equiv B \to C$, and $\Gamma, x : B \triangleright x : B$ holds by Lemma 2; $\Gamma, x : B \triangleright Mx : C$ and $\Gamma \triangleright \lambda x : B.Mx : B \to C$ by definition.
- $M \xrightarrow{\widehat{\eta}} \lambda P : B.MP$. In this case, also $A \equiv B \to C$, and from $P : B \triangleright P : B$, we obtain $\Gamma, P : B \triangleright P : B$ by Lemma 2; $\Gamma, P : B \triangleright MP : C$ and $\Gamma \triangleright \lambda P : B.MP : B \to C$ by definition.

Proof of Theorem 1 □

Another important property of these systems is that the *sp*-rule allows for the specification of terms by adding more structure to them. This can be expressed in the following way.

Lemma 5 *If $\Gamma \triangleright M : A$, then there is a term N such that $M \xrightarrow{sp}{}^{*} N$ and $N :: A$.*

Proof of Lemma 5 By definition of the structure of the type A.

Proof of Lemma 5 □

The adequacy of these expansion rules for the associated extensional equalities emerges from the following theorems, which state that the limitations imposed on the reduction systems do not cause a reduction in valid equality, as they are introduced exactly to avoid reduction loops.

Lemma 6 *The relation* $\underset{lw}{\longleftrightarrow}^*$ *is closed by context.*

Proof of Lemma 6 To show that $\underset{lw}{\longleftrightarrow}^*$ is stable by context closure (i.e., $M_1 \underset{lw}{\longleftrightarrow}^* M_2$ implies $C[M_1] \underset{lw}{\longleftrightarrow}^* C[M_2]$ for any context $C[]$), it is sufficient to show the property for the relation $\underset{lw}{\longleftrightarrow}$. According to Definition 11, there are only two cases where this is not immediately evident:

- If $M \xrightarrow{\widehat{\eta}} (\lambda P : A.MP)$, then the relation is closed for all contexts *except* in the case where the terms M and $(\lambda P : A.MP)$ are applied to a term L that is fully specified. In this case, it is clear by Lemma 3 that there is a substitution σ such that $\mathrm{Match}(L, P) = \sigma$ and $\sigma(P) = L$, and since $Var(P) \cap FV(M) = \emptyset$ holds by definition, then $M[\sigma] = M$ and
$$(\lambda P : A.MP)L \xrightarrow{\beta} (MP)[\sigma] = (M[\sigma]P[\sigma]) = ML$$
- If $M \xrightarrow{sp} \langle \pi_1(M), \pi_2(M)\rangle$, then the relation is closed for all the contexts *except* in the case where the terms M and $\langle \pi_1(M), \pi_2(M)\rangle$ are projected. In this case, the rules π_i can be used as follows:
$$\pi_i(M) \xleftarrow{\pi_i} \pi_i(\langle \pi_1(M), \pi_2(M)\rangle)$$

Proof of Lemma 6 □

Lemma 7 *The relation* $\underset{lwp}{\longleftrightarrow}^*$ *is closed by context.*

Proof of Lemma 7 The proof proceeds as in Lemma 6 except for the case $M \xrightarrow{\eta} (\lambda x : A.Mx)$, for which $(\lambda x : A.Mx)L \xrightarrow{\beta} ML$, concluding the proof.

Proof of Lemma 7 □

Now, the adequacy of both extensional reduction systems can be shown in the following way.

Theorem 2 (Equivalence of $\underset{lw}{\longleftrightarrow}^*$ and $\doteq$) *$M \doteq N$ if and only if $M \underset{lw}{\longleftrightarrow}{}^* N$.*

Proof of Theorem 2 (Equivalence of $\underset{lw}{\longleftrightarrow}^*$ and $\doteq$) If $M \underset{lw}{\longleftrightarrow}{}^* N$, then $M \underset{lw}{\longleftrightarrow} M_1 \underset{lw}{\longleftrightarrow} \ldots \underset{lw}{\longleftrightarrow} M_n \underset{lw}{\longleftrightarrow} N$, which implies $M \doteq M_1 \doteq \ldots \doteq M_n \doteq N$ because $\underset{lw}{\longleftrightarrow}$ 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 $M \underset{lw}{\longleftrightarrow}{}^* N$, 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 $M \underset{lw}{\longleftrightarrow}{}^* M$ is trivial. Otherwise, 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 follow by the induction hypothesis.
- If $M \doteq N$ comes from an equality $M' \doteq N'$ by a context closure, then by the induction hypothesis $M' \underset{lw}{\longleftrightarrow}{}^* N'$, and by Lemma 6, $M \underset{lw}{\longleftrightarrow}{}^* N$.
- If $M \doteq N$ comes from an axiom, then one must show how to obtain $M \underset{lw}{\longleftrightarrow}{}^* N$ explicitly. Only the problematic cases are listed here:
  1. $\lambda P : A.M =_{\eta_p} \lambda Q : A.(\lambda P : A.M)Q$. The cases to consider are as follows:
     (a) P is fully specified by A.
       – Q is fully specified by A. Then necessarily, $P =_\alpha Q$, 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 only specified by A by definition of η_p). By Lemma 3, Match$(P, Q) = \sigma$ and $\sigma(Q) = P$, so

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

(b) P is not fully specified by A.

– Q is fully specified by A. Then

$$\lambda P : A.M \xrightarrow{\widehat{\eta}} \lambda Q : A.(\lambda P : A.M)Q$$

– Q is not fully specified by A. Take $R :: A$ (which exists by definition of full specification) and let σ be the substitution such that $\text{Match}(R, Q) = \sigma$ and $\sigma(Q) = R$ (which exists by Lemma 3). Then,

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

2. $\langle M, N\rangle =_{sp} \langle \pi_1(\langle M, N\rangle), \pi_2(\langle M, N\rangle)\rangle$. An expansion cannot be applied to a term that is a pair, but the reductions π_i may be used as follows:

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

Proof of Theorem 2 □

Theorem 3 (Equivalence of $\underset{lwp}{\longleftrightarrow}^*$ and $\doteq$) *$M \doteq N$ if and only if $M \underset{lwp}{\longleftrightarrow}{}^* N$.*

Proof of Theorem 3 (Equivalence of $\underset{lwp}{\longleftrightarrow}^*$ and $\doteq$) The only difference from the proof of Theorem 2 comes when one must show that $M \doteq N$ (from an axiom) implies $M \underset{lwp}{\longleftrightarrow}{}^* N$. Take the problematic case of $M = \lambda P : A.M =_{\eta_p} \lambda Q : A.(\lambda P : A.M)Q = N$. Let z be a variable. By Lemma 5, $z \xrightarrow{sp}{}^* Z$, where Z is fully specified by its type. By definition of η_p the pattern Q is specified by A, and so by Lemma 3 there is a substitution σ such that $\text{Match}(Z, Q) = \sigma$ and $\sigma(Q) = Z$. Thus, the following is obtained:

$$\begin{array}{ccc}
 & & \lambda Q : A.(\lambda P : A.M)Q \\
 & & \downarrow_{\eta} \\
\lambda P : A.M & & \lambda z : A.(\lambda Q : A.(\lambda P : A.M)Q)z \\
\downarrow_{\eta} & & \downarrow_{*}^{sp} \\
\lambda z : A.(\lambda P : A.M)z & & \lambda z : A.(\lambda Q : A.(\lambda P : A.M)Q)Z \\
\downarrow_{*}^{sp} & & \downarrow_{\beta} \\
\lambda z : A.(\lambda P : A.M)Z & = & \lambda z : A.(\lambda P : A.M)Z
\end{array}$$

Proof of Theorem 3 $\square$

Coming back to Example 1, the following three terms can be proved to be equal using the reduction relations $\xrightarrow{lw}$ and $\xrightarrow{lwp}$:

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

Indeed, the last two terms reduce to the first term via the $\xrightarrow{lw}$ reduction relation, while all terms reduce to $\lambda x : A \times B.\langle \pi_1(x), \langle \pi_1(x), \pi_2(x)\rangle\rangle$ via the $\xrightarrow{lwp}$ relation.

This example clearly illustrates the redundancy of layered and wildcard patterns by means of the $\xrightarrow{lw}$ reduction system: indeed, every pattern abstraction $\lambda P : A.M$ can be rewritten via the $\xrightarrow{lw}$ reduction relation to an equivalent function $\lambda P' : A.M'$ such that P' contains only variables and pairs. This corresponds to the fact that contraction and weakening rules are redundant in intuitionist logic [Gal93]. At the same time, this provides another possible reading: removing (at compile time) wildcard and layered patterns from λ-abstractions using the reduction system could lead to program optimizations that will then be interpreted by simpler computation rules. In fact, the pattern-matching function becomes simpler.

Theorem 4 (Redundancy in System $\xrightarrow{lw}$) *Any term M can be lw-reduced to a term M' that contains no layered and no wildcard patterns.*

Proof of Theorem 4 (Redundancy in System $\xrightarrow{lw}$) By induction on the structure of the term M, it can be shown that for every term M there is an M' such that $M \xrightarrow{lw}{}^{*} M'$, M' contains no layered and no wildcard patterns, and the reduction sequence $M \xrightarrow{lw}{}^{*} M'$ does not contain root expansions if M is not a λ-abstraction.

- If $M = x$, the property is trivial.
- If $M = \langle M_1, M_2\rangle$, the property holds by induction hypothesis.
- If $M = \lambda P : A.N$, then N reduces to a term N' with no layered and no wildcard patterns and so $\lambda P : A.N \xrightarrow{lw}{}^{*} \lambda P : A.N'$. If P contains no

layered and no wildcard patterns, then $M' = \lambda P : A.N'$ and the proof is complete by induction hypothesis. Otherwise, an expansion can be applied because P is not fully specified and then $\lambda P : A.N' \xrightarrow{\widehat{\eta}} \lambda Q : A.(\lambda P : A.N')Q$, for some fully specified pattern Q. By Lemma 3, there is a substitution σ such that $\mathrm{Match}(Q, P) = \sigma$, and so $\lambda Q : A.(\lambda P : A.N')Q \xrightarrow{\beta} \lambda Q : A.N'[\sigma] = M'$. This concludes the proof of this case, because

- N' does not contain layered and wildcard patterns by induction hypothesis,
- Q does not contain layered and wildcard patterns by definition, and
- the substitution σ affects only variables of N' using subterms of Q.

- If $M = \pi_i(N)$, then by induction hypothesis N reduces to a term N' with no layered and no wildcard patterns, and there are no root expansions in the reduction sequence because N is of product type and cannot be a λ-abstraction. As a consequence, $M = \pi_i(N) \xrightarrow{lw}{}^* \pi_i(N') = M'$, and the proof is complete.
- If $M = NL$, two different cases are considered. If N is not a λ-abstraction, then by induction hypothesis there are N' and L' such that $N \xrightarrow{lw}{}^* N'$, $L \xrightarrow{lw}{}^* L'$, N' and L' do not contain layered and wildcard patterns, and the sequence $N \xrightarrow{lw}{}^* N'$ does not contain root expansions. Both sequences $N \xrightarrow{lw}{}^* N'$ and $L \xrightarrow{lw}{}^* L'$ can be combined into a reduction sequence $M = NL \xrightarrow{lw}{}^* N'L' = M'$ without root expansions, and the property holds because M' does not contain layered and wildcard patterns. If N is a λ-abstraction $\lambda P.K$, then by induction hypothesis there exist reduction sequences $K \xrightarrow{lw}{}^* K'$ and $L \xrightarrow{lw}{}^* L'$ such that K' and L' do not contain layered and wildcard patterns. Moreover, by Lemma 5, L' can be further *sp*-reduced to obtain a fully specified term L'' that will match the pattern P by Lemma 3. Let σ be the substitution obtained by $\mathrm{Match}(L'', P)$. The following reduction sequence can be built:

$$M = (\lambda P.K)L \xrightarrow{lw}{}^* (\lambda P.K')L' \xrightarrow{sp}{}^* (\lambda P.K')L'' \xrightarrow{\beta} K'[\sigma] = M'$$

Notice that this sequence does not contain root expansions, and that M' verifies the property because

- K' does not contain layered and wildcard patterns by induction hypothesis,
- L'' does not contain layered and wildcard patterns because it is obtained by *sp*-reduction from L' which does not contain layered and wildcard patterns by induction hypothesis, and
- the substitution σ affects only variables of K' using subterms of L''.

Proof of Theorem 4 $\square$

Returning again to Example 1, the three terms can be checked to verify the property stated in Theorem 4. 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 $\lambda x@\langle y, \_\rangle : A \times B.\langle y, x\rangle$ the following reduction sequence must be performed:

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

Now, the redundancy of layered, wildcard, and product patterns can be stated by means of the reduction system $\xrightarrow{lwp}$: indeed, every pattern abstraction $\lambda P : A.M$ can be rewritten via the $\xrightarrow{lwp}$ reduction relation to an equivalent function $\lambda x : A.M'$.

Theorem 5 (Redundancy in System $\xrightarrow{lwp}$) *Any term M can be lwp-reduced to a term M' that contains no complex patterns.*

Proof of Theorem 5 (Redundancy in System $\xrightarrow{lwp}$) The proof proceeds by induction on the structure of the term M as in Theorem 4. The only different case with respect to the previous proof is when $M = \lambda P : A.N$. By induction hypothesis the property holds for N, and so there is a term N' without complex patterns such that $N \xrightarrow{lwp}{}^* N'$. If P is a variable x, then $\lambda x : A.N \xrightarrow{lwp}{}^* \lambda x : A.N' = M'$, which concludes the proof. Otherwise,

$M = \lambda P : A.N \xrightarrow{lwp}{}^{*} \lambda P : A.N' \xrightarrow{\eta} \lambda z : A.(\lambda P : A.N')z$, and z *sp*-reduces by Lemma 5 to a fully specified term Z. By Lemma 3 there is a substitution σ such that $\mathrm{Match}(Z, P) = \sigma$, and so $\lambda z : A.(\lambda P : A.N')z \xrightarrow{sp}{}^{*} \lambda z : A.(\lambda P : A.N')Z \xrightarrow{\beta} \lambda z : A.N'[\sigma] = M'$, which concludes the proof because

- N' does not contain complex patterns by induction hypothesis,
- Z does not contain layered and wildcard patterns because it is obtained from a variable by *sp*-reduction steps, and
- the substitution σ affects only term variables of N' using subterms of Z.

Proof of Theorem 5 □

Coming back to Example 1, the term $\lambda x : A \times B.\langle \pi_1(x), \langle \pi_1(x), \pi_2(x)\rangle\rangle$, which has no complex patterns, is a common reduct of terms (1), (2), and (3).

4 Modular Properties of $\xrightarrow{lw}$ and $\xrightarrow{lwp}$

In this section, it is shown that both relations $\xrightarrow{lw}$ and $\xrightarrow{lwp}$ are confluent and strongly normalizing on the set of typed terms. Standard techniques for showing confluence and strong normalization for the typed λ-calculus will not work directly because of the extensional expansions. To overcome this annoyance, divide-and-conquer techniques [Aka93] are used, allowing the desired properties to be shown.

Definition 12 (Preservation of Normal Forms) *Let $\langle \mathcal{A}, \xrightarrow{R}\rangle$ and $\langle \mathcal{A}, \xrightarrow{S}\rangle$ be two abstract reduction systems. Relation $\xrightarrow{R}$ preserves S-normal forms if $\xrightarrow{R}$ does not create S-redexes.*

The modular techniques used here to show confluence and strong normalization can be summarized by the following theorems.

Theorem 6 (Modular Confluence [Aka93]) *Let $\langle \mathcal{A}, \xrightarrow{R}\rangle$ and $\langle \mathcal{A}, \xrightarrow{S}\rangle$ be two abstract reduction systems such that*

- $\xrightarrow{R}$ *is confluent,*
- $\xrightarrow{S}$ *is confluent and strongly normalizing, and*
- $\forall M, N \in \mathcal{A}\ \ M \xrightarrow{R} N$ *implies* $M^S \xrightarrow{R}{}^+ N^S$.

Then the relation $\xrightarrow{R} \cup \xrightarrow{S}$ *is confluent.*

Theorem 7 (Modular Strong Normalization [Aka93]) *Let* $\langle \mathcal{A}, \xrightarrow{R} \rangle$ *and* $\langle \mathcal{A}, \xrightarrow{S} \rangle$ *be two abstract reduction systems such that*

- $\xrightarrow{R}$ *is strongly normalizing,*
- $\xrightarrow{S}$ *is confluent and strongly normalizing, and*
- $\forall M, N \in \mathcal{A}\ \ M \xrightarrow{R} N$ *implies* $M^S \xrightarrow{R}{}^+ N^S$.

Then the relation $\xrightarrow{R} \cup \xrightarrow{S}$ *is strongly normalizing.*

The reduction relations are now separated into three different subrelations: $\xrightarrow{B}$ denotes the relation generated only by $\xrightarrow{\beta,\pi_1,\pi_2}$, $\xrightarrow{E}$ denotes the relation generated only by $\xrightarrow{\eta,sp}$, and $\xrightarrow{\widehat{E}}$ denotes the relation generated only by $\xrightarrow{\widehat{\eta},sp}$. The next sections discuss the techniques for proving all of the properties that are required as hypotheses in the previous theorems.

4.1 Properties of the Nonextensional Rules

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

- $M \triangleright M$.
- If $M \triangleright M'$, then $\pi_1(\langle M, N\rangle) \triangleright M'$, $\pi_2(\langle N, M\rangle) \triangleright M'$, $\lambda P.M \triangleright \lambda P.M'$, and $\pi_i(M) \triangleright \pi_i(M')$.

- If $M \triangleright M'$ and $N \triangleright N'$, then $\langle M, N\rangle \triangleright \langle M', N'\rangle$, $MN \triangleright M'N'$, and $(\lambda P : A.N)M \triangleright N'[\text{Match}(M', P)]$ if $\text{Match}(M, P)$ is defined.

Parallel reduction is stable by substitution and it plays an important role in the confluence proof.

Lemma 8 *If $M \triangleright M'$, $N \triangleright N'$, and $Match(N, P)$ is defined, $M[Match(N, P)] \triangleright M'[Match(N', P)]$.*

Proof of Lemma 8 This is a simple proof by induction on the structure of M using the following facts:

- If $\text{Match}(M, P) = \theta$ and $M \triangleright M'$, then there is a substitution θ' such that $\text{Match}(M', P) = \theta'$ and $\theta(x) \triangleright \theta'(x)$ for all x in $\text{Dom}(\theta)$.
- If P and Q have no common variables and $Var(P) \cap FV(N) = \emptyset$, then if $\text{Match}(M, P) = \theta_1$ and $\text{Match}(N, Q) = \theta_2$,

$$L[\theta_2][\text{Match}(M[\theta_2], P)] = L[\theta_1][\theta_2]$$

Proof of Lemma 8 $\square$

The confluence of $\xrightarrow{B}$ for the set of all the terms of the LWPP calculus can now be stated.

Theorem 8 (Confluence of $\xrightarrow{B}$) *The reduction relation $\xrightarrow{B}$ is confluent.*

Proof of Theorem 8 (Confluence of $\xrightarrow{B}$) By Tait-Martin Löf's technique, the following three properties are needed:

- The relation $\triangleright$ satisfies the the diamond property.

  Subproof: By case analysis, using Lemma 8.
- A relation satisfying the diamond property is confluent.

  Subproof: See [Bar84].
- The reflexive and transitive closures of $\triangleright$ and $\xrightarrow{B}$ are the same relation.

  Subproof: The inclusion $\triangleright^* \subseteq \xrightarrow{B}^*$ follows from $\triangleright \subseteq \xrightarrow{B}^*$, that follows by induction on the definition of $\triangleright$. The inclusion $\xrightarrow{B}^* \subseteq \triangleright^*$ follows from $\xrightarrow{B} \subseteq \triangleright$ by induction on the structure of terms, then by cases.

The confluence property of the relation $\xrightarrow{B}$ can now be derived from the confluence property of the relation $\triangleright$.

Proof of Theorem 8 $\square$

Theorem 9 (Strong Normalization of $\xrightarrow{B}$) *The relation $\xrightarrow{B}$ is strongly normalizing on typed terms.*

Proof of Theorem 9 (Strong Normalization of $\xrightarrow{B}$) Since for every term M in the LWPP calculus there is a translated term $\mathcal{S}(M)$ in the TPC such that $M \xrightarrow{B} M'$ implies $\mathcal{S}(M) \xrightarrow{\mathrm{TPC}}{}^{+}\mathcal{S}(M')$ (see Lemma 14 in the Appendix), and the relation $\xrightarrow{\mathrm{TPC}}$ is strongly normalizing for all the typed terms in the TPC [KPT96], it can be concluded that $\xrightarrow{B}$ is strongly normalizing.

Proof of Theorem 9 $\square$

4.2 Properties of the Extensional Rules

This section clarifies that the reduction relations $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$ are strongly normalizing and confluent. The confluence property is shown by Newman's lemma, that is, by the local confluence and the strong normalization properties. For the strong normalization property, the proof of [Min79] is formalized for the typed λ-calculus: to every reduction relation $C \in \{E, \widehat{E}\}$ and to every term M, a natural number $\mathcal{E}_C(M)$ is assigned and it is proven that $M \xrightarrow{C} M'$ implies $\mathcal{E}_C(M) > \mathcal{E}_C(M')$. This shows that $\mathcal{E}_C(M)$ is an upper bound for the length of any C-reduction sequence.

4.2.1 A Decreasing Measure Associated with the Extensional Rules

The number of constructors and base types in the type A is noted as $|A|$:

$$
\begin{array}{ll}
|A| = 1 & A \text{ is a base type} \\
|A_1 \times A_2| = |A_1 \to A_2| = 1 + |A_1| + |A_2| & \text{otherwise}
\end{array}
$$

The set of *positions* of the term M is denoted by $\mathcal{P}(M)$, and the subterm of M at position u is denoted by M/u. Both notions are defined by induction as follows:

$$\begin{array}{lll}
\mathcal{P}(x) & = \{\epsilon\} & \\
\mathcal{P}(M_1 M_2) & = \mathcal{P}(\langle M_1, M_2\rangle) = \{\epsilon\} \cup 1.\mathcal{P}(M_1) \cup 2.\mathcal{P}(M_2) & \\
\mathcal{P}(\lambda P : A.M) & = \mathcal{P}(\pi_i(M)) & = \{\epsilon\} \cup 1.\mathcal{P}(M)
\end{array}$$

where $u.S = \{u.v \mid v \in S\}$ and $u.\epsilon = u$.

$$\begin{array}{lll}
M/\epsilon & = M & \\
(M_1 M_2)/1.v & = \langle M_1, M_2\rangle/1.v = M_1/v & \\
(M_1 M_2)/2.v & = \langle M_1, M_2\rangle/2.v = M_2/v & \\
\lambda P : A.M/1.v & = \pi_i(M)/1.v & = M/v
\end{array}$$

Now, for every reduction relation $C \in \{E, \widehat{E}\}$, every term M, and every $u \in \mathcal{P}(M)$, $\mathcal{D}_C(M,u)$ is defined as follows:

$$\mathcal{D}_C(M,u) = \begin{cases} 2^{\Sigma_{1 \le i \le n} |A_i|} & \text{if } M/u \text{ is a } C\text{-redex} \\ 0 & \text{otherwise} \end{cases}$$

where $A_1 \ldots A_n$ are the types of C-redexes in M that contain M/u (including M/u and M).

Example 5 Let A and B be two base types, z a variable of type $A \times B \to A \times B$, and $M = \lambda x : A \times B.zx$. The terms $M/\epsilon = M$ and $M/1.1 = z$ are $\widehat{E}$-redexes but not E-redexes, while $M/1 = zx$ and $M/1.2 = x$ are $\widehat{E}$-redexes and also E-redexes. As a consequence,

$$\begin{array}{llll}
\mathcal{D}_E(M,\epsilon) = 0 & \mathcal{D}_E(M,1) = 2^3 & \mathcal{D}_E(M,1.1) = 0 & \mathcal{D}_E(M,1.2) = 2^6 \\
\mathcal{D}_{\widehat{E}}(M,\epsilon) = 2^7 & \mathcal{D}_{\widehat{E}}(M,1) = 2^{10} & \mathcal{D}_{\widehat{E}}(M,1.1) = 2^{17} & \mathcal{D}_{\widehat{E}}(M,1.2) = 2^{13}
\end{array}$$

End of Example 5 □

To every $C \in \{E, \widehat{E}\}$ and to every term M the natural number $\mathcal{E}_C(M)$ is associated, which is defined as

$$\mathcal{E}_C(M) = \sum \{\mathcal{D}_C(M,u) \mid u \in \mathcal{P}(M) \text{ and } M/u \text{ is a } C\text{-redex}\}$$

Example 6 For $M = \lambda x : A \times B.zx$ as in Example 5,

$$\mathcal{E}_E(M) = 2^3 + 2^6 \quad \text{and} \quad \mathcal{E}_{\widehat{E}}(M) = 2^7 + 2^{10} + 2^{17} + 2^{13}$$

End of Example 6 □

To prove that $M \xrightarrow{C} M'$ implies $\mathcal{E}_C(M) > \mathcal{E}_C(M')$, it would be interesting to proceed by induction on the structure of the term M. As $\mathcal{E}_C(M)$ cannot always be easily expressed in terms of the function $\mathcal{E}_C$ itself, a new measure is introduced, called $\mathcal{F}_C$, having the following properties:

- $\mathcal{E}_C(M)$ can be defined in terms of $\mathcal{E}_C$ and/or $\mathcal{F}_C$ applied to the subterms of M, and
- if M reduces to M' by an internal C-step, then $\mathcal{F}_C(M) > \mathcal{F}_C(M')$.

These properties will be sufficient to conclude with the strong normalization result for the relations E and $\widehat{E}$. The decreasing measure $\mathcal{F}_C$ is as follows: for every reduction relation $C \in \{E, \widehat{E}\}$, every term M, and every $u \in \mathcal{P}(M)$ such that $u \neq \epsilon$, $\mathcal{B}_C(M, u)$ is defined as

$$\mathcal{B}_C(M,u) = \begin{cases} 2^{\Sigma_{1\leq i\leq n}|A_i|} & \text{if } M/u \text{ is a } C\text{-redex} \\ 0 & \text{otherwise} \end{cases}$$

where $A_1 \dots A_n$ are the types of C-redexes in M that are different from M and contain M/u.

Example 7 For $M = \lambda x : A \times B.zx$ as in Example 5,

$$\begin{array}{lll} \mathcal{B}_E(M,1) = 2^3 & \mathcal{B}_E(M,1.1) = 0 & \mathcal{B}_E(M,1.2) = 2^6 \\ \mathcal{B}_{\widehat{E}}(M,1) = 2^3 & \mathcal{B}_{\widehat{E}}(M,1.1) = 2^{10} & \mathcal{B}_{\widehat{E}}(M,1.2) = 2^6 \end{array}$$

End of Example 7 □

To every $C \in \{E, \widehat{E}\}$ and to every term M, the natural number $\mathcal{F}_C(M)$ is associated, which is defined as

$$\mathcal{F}_C(M) = \sum \{\mathcal{B}_C(M,u) \mid \epsilon \neq u \in \mathcal{P}(M) \text{ and } M/u \text{ is a } C\text{-redex}\}$$

Example 8 For $M = \lambda x : A \times B.zx$ as in Example 5,

$$\mathcal{F}_E(M) = 2^3 + 2^6 \quad \text{and} \quad \mathcal{F}_{\widehat{E}}(M) = 2^3 + 2^{10} + 2^6$$

End of Example 8 □

Fact 1 *For every reduction relation $C \in \{E, \widehat{E}\}$, if M is a C-redex, then $\mathcal{E}_C(M) = \mathcal{D}_C(M, \epsilon) + \mathcal{D}_C(M, \epsilon) * \mathcal{F}_C(M)$; otherwise $\mathcal{E}_C(M) = \mathcal{F}_C(M)$.*

Indeed, coming back to Example 5,

$$
\begin{aligned}
2^3 + 2^6 = \mathcal{E}_E(\lambda x : A \times B.zx) &= \mathcal{F}_E(\lambda x : A \times B.zx) = 2^3 + 2^6 \\
2^7 + 2^{10} + 2^{17} + 2^{13} &= \mathcal{E}_{\widehat{E}}(M) \\
&= \mathcal{D}_{\widehat{E}}(M, \epsilon) * (1 + \mathcal{F}_{\widehat{E}}(M)) \\
&= 2^7 * (1 + 2^3 + 2^{10} + 2^6)
\end{aligned}
$$

Corollary 1 *For every reduction relation $C \in \{E, \widehat{E}\}$ and every term M such that $\Gamma \triangleright M : A$, $2^{|A|} \geq \mathcal{D}_C(M, \epsilon)$ and so $2^{|A|} * (1 + \mathcal{F}_C(M)) \geq \mathcal{E}_C(M)$.*

Lemma 9 (Properties of the Measure)

- $\mathcal{E}_C(\langle M, N\rangle) = \mathcal{E}_C(M) + \mathcal{E}_C(N)$.
- $\mathcal{F}_C(\pi_i(M)) = \mathcal{F}_C(M)$, *for* $i = 1, 2$.
- $\mathcal{F}_C(\lambda P : A.M) = \mathcal{E}_C(M)$.
- $\mathcal{F}_E(MN) = \mathcal{F}_E(M) + \mathcal{E}_E(N)$.
- $\mathcal{F}_{\widehat{E}}(MN) = \begin{cases} \mathcal{F}_{\widehat{E}}(M) + \mathcal{E}_{\widehat{E}}(N) & \text{if } N \text{ is fully specified,} \\ \mathcal{E}_{\widehat{E}}(M) + \mathcal{E}_{\widehat{E}}(N) & \text{if } N \text{ is not fully specified.} \end{cases}$

Remark 1 *If $M \xrightarrow{C} M'$ is an internal reduction step, then M is a C-redex if and only if M' is a C-redex, and so $\mathcal{D}_C(M, \epsilon) = \mathcal{D}_C(M', \epsilon)$. As a consequence, from Fact 1, $\mathcal{E}_C(M) > \mathcal{E}_C(M')$ if and only if $\mathcal{F}_C(M) > \mathcal{F}_C(M')$.*

Using all the previous properties, the desired result is shown as follows.

Theorem 10 *The relations $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$ are strongly normalizing.*

Proof of Theorem 10 It can be shown that $M \xrightarrow{C} M'$ implies $\mathcal{E}_C(M) > \mathcal{E}_C(M')$, so that $\mathcal{E}_C(M)$ is an upper bound for the length of any C-reduction sequence. To show this, it is possible to simultaneously verify that whenever $M \xrightarrow{C} M'$ is an internal C-reduction step, then $\mathcal{F}_C(M) > \mathcal{F}_C(M')$.

Suppose that $M \xrightarrow{C} M'$. Proceed by case analysis, according to the nature of the reduction step $\xrightarrow{C}$. First consider all the cases where $\xrightarrow{C}$ is an external C-reduction step, so that it is only necessary to check that $\mathcal{E}_C(M) > \mathcal{E}_C(M')$.

- $M \xrightarrow{sp} \langle \pi_1(M), \pi_2(M)\rangle$, for M of type $A \times B$. Then for $C \in \{E, \widehat{E}\}$:

$$\begin{array}{ll}
\mathcal{E}_C(M) & =_{\text{Fact 1}} \\
\mathcal{D}_C(M,\epsilon) * (1 + \mathcal{F}_C(M)) & = \\
2^{|A \times B|} * (1 + \mathcal{F}_C(M)) & = \\
2^{|A|} * 2^{|B|} * 2 * (1 + \mathcal{F}_C(M)) & > \\
(2^{|A|} + 2^{|B|}) * (1 + \mathcal{F}_C(M)) & = \\
2^{|A|} * (1 + \mathcal{F}_C(M)) + 2^{|B|} * (1 + \mathcal{F}_C(M)) & =_{\text{Lemma 9}} \\
2^{|A|} * (1 + \mathcal{F}_C(\pi_1(M))) + 2^{|B|} * (1 + \mathcal{F}_C(\pi_2(M))) & \geq_{\text{Corollary 1}} \\
\mathcal{E}_C(\pi_1(M)) + \mathcal{E}_C(\pi_2(N)) & =_{\text{Lemma 9}} \\
\mathcal{E}_C(\langle \pi_1(M), \pi_2(M)\rangle) &
\end{array}$$

- $M \xrightarrow{\eta} \lambda x : A.Mx$, for M of type $A \to B$:

$$\begin{array}{ll}
\mathcal{E}_E(M) & =_{\text{Fact 1}} \\
\mathcal{D}_E(M,\epsilon) * (1 + \mathcal{F}_E(M)) & = \\
2^{|A \to B|} * (1 + \mathcal{F}_E(M)) & = \\
2^{|A|} * 2^{|B|} * 2 * (1 + \mathcal{F}_E(M)) & > \\
2^{|B|} * (1 + 2^{|A|} + \mathcal{F}_E(M)) & \geq_{\text{Corollary 1}} \\
2^{|B|} * (1 + \mathcal{D}_E(x,\epsilon) + \mathcal{F}_E(M)) & \geq_{\text{Definition of } \mathcal{E}_E(x)} \\
2^{|B|} * (1 + \mathcal{E}_E(x) + \mathcal{F}_E(M)) & =_{\text{Lemma 9}} \\
2^{|B|} * (1 + \mathcal{F}_E(Mx)) & \geq_{\text{Corollary 1}} \\
\mathcal{E}_E(Mx) & =_{\text{Fact 1 and Lemma 9}} \\
\mathcal{E}_E(\lambda x : A.Mx) &
\end{array}$$

- $M \xrightarrow{\widehat{\eta}} \lambda P : A.MP$, where M is of type $A \to B$ and $A_1 \ldots A_k$ are the types of the (disjoint) $\widehat{E}$-redexes in P.

Remark 2 *Let P be a term that is at the same time a fully specified pattern of type A (as the term P introduced by the expansion rule $\xrightarrow{\widehat{\eta}}$). Let $Func(P)$ be the* multiset *of types of the functional variables of P. Then every variable in $Func(P)$ is an $\widehat{E}$-redex, and since these $\widehat{E}$-redexes are necessarily disjoint, then $\mathcal{E}_{\widehat{E}}(P) = 2^{\Sigma_{A_i \in Func(P)}|A_i|}$. Notice also that $|A| \geq \Sigma_{A_i \in Func(P)}|A_i|$.*

$$
\begin{array}{ll}
\mathcal{E}_{\widehat{E}}(M) & =_{\text{Fact 1}} \\
\mathcal{D}_{\widehat{E}}(M,\epsilon) * (1 + \mathcal{F}_{\widehat{E}}(M)) & = \\
2^{|A \to B|} * (1 + \mathcal{F}_{\widehat{E}}(M)) & = \\
2^{|A|} * 2^{|B|} * 2 * (1 + \mathcal{F}_{\widehat{E}}(M)) & > \\
2^{|B|} * (1 + 2^{|A|} + \mathcal{F}_{\widehat{E}}(M)) & \geq_{\text{Remark 2}} \\
2^{|B|} * (1 + 2^{\Sigma_{A_i \in Func(P)}|A_i|} + \mathcal{F}_{\widehat{E}}(M)) & =_{\text{Remark 2}} \\
2^{|B|} * (1 + \mathcal{E}_{\widehat{E}}(P) + \mathcal{F}_{\widehat{E}}(M)) & =_{\text{Lemma 9}} \\
2^{|B|} * (1 + \mathcal{F}_{\widehat{E}}(MP)) & \geq_{\text{Corollary 1}} \\
\mathcal{E}_{\widehat{E}}(MP) & =_{\text{Lemma 9}} \\
\mathcal{E}_{\widehat{E}}(\lambda P : A.MP) &
\end{array}
$$

Now consider the cases where $M \xrightarrow{C} M'$ is an internal C-reduction step. By Remark 1, it is sufficient to show that $\mathcal{F}_C(M) > \mathcal{F}_C(M')$. To do this, proceed by induction on the structure of M.

- $\langle L, N\rangle \xrightarrow{C} \langle L', N\rangle$, where $L \xrightarrow{C} L'$ (the case $N \xrightarrow{C} N'$ being similar).

$$
\begin{array}{ll}
\mathcal{F}_C(\langle L, N\rangle) & =_{\text{Fact 1 and Lemma 9}} \\
\mathcal{E}_C(L) + \mathcal{E}_C(N) & >_{\text{i.h.}} \\
\mathcal{E}_C(L') + \mathcal{E}_C(N) & =_{\text{Fact 1 and Lemma 9}} \\
\mathcal{F}_C(\langle L', N\rangle) &
\end{array}
$$

- $\pi_i(L) \xrightarrow{C} \pi_i(L')$, where $L \xrightarrow{C} L'$. Notice that in this case $L \xrightarrow{C} L'$ must be an internal reduction, and so by induction hypothesis $\mathcal{F}_C(L) > \mathcal{F}_C(L')$.

$$
\begin{array}{ll}
\mathcal{F}_C(\pi_i(L)) & =_{\text{Lemma 9}} \\
\mathcal{F}_C(L) & >_{\text{i.h.}} \\
\mathcal{F}_C(L') & =_{\text{Lemma 9}} \\
\mathcal{F}_C(\pi_i(L')) &
\end{array}
$$

- $\lambda P.L \xrightarrow{C} \lambda P.L'$, where $L \xrightarrow{C} L'$.

$$\begin{array}{ll} \mathcal{F}_C(\lambda P.L) & =_{\text{Lemma 9}} \\ \mathcal{E}_C(L) & >_{\text{i.h.}} \\ \mathcal{E}_C(L') & =_{\text{Lemma 9}} \\ \mathcal{F}_C(\lambda P.L') & \end{array}$$

- $LN \xrightarrow{E} L'N$, where $L \xrightarrow{E} L'$. Notice that also in this case $L \xrightarrow{E} L'$ must be an internal reduction, and so by induction hypothesis $\mathcal{F}_E(L) > \mathcal{F}_E(L')$.

$$\begin{array}{ll} \mathcal{F}_E(LN) & =_{\text{Lemma 9}} \\ \mathcal{F}_E(L) + \mathcal{E}_E(N) & >_{\text{i.h.}} \\ \mathcal{F}_E(L') + \mathcal{E}_E(N) & =_{\text{Lemma 9}} \\ \mathcal{F}_E(L'N) & \end{array}$$

- $LN \xrightarrow{E} LN'$, where $N \xrightarrow{E} N'$.

$$\begin{array}{ll} \mathcal{F}_E(LN) & =_{\text{Lemma 9}} \\ \mathcal{F}_E(L) + \mathcal{E}_E(N) & >_{\text{i.h.}} \\ \mathcal{F}_E(L) + \mathcal{E}_E(N') & =_{\text{Lemma 9}} \\ \mathcal{F}_E(LN') & \end{array}$$

- $LN \xrightarrow{\widehat{E}} L'N$, where $L \xrightarrow{\widehat{E}} L'$. There are two cases for N:

  1. N is fully specified. In this case, $L \xrightarrow{\widehat{E}} L'$ must be an internal $\widehat{E}$-reduction step.

$$\begin{array}{ll} \mathcal{F}_{\widehat{E}}(LN) & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N) & >_{\text{i.h. and Remark 1}} \\ \mathcal{F}_{\widehat{E}}(L') + \mathcal{E}_{\widehat{E}}(N) & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(L'N) & \end{array}$$

  2. N is not fully specified:

$$\begin{array}{ll} \mathcal{F}_{\widehat{E}}(LN) & =_{\text{Lemma 9}} \\ \mathcal{E}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N) & >_{\text{i.h.}} \\ \mathcal{E}_{\widehat{E}}(L') + \mathcal{E}_{\widehat{E}}(N) & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(L'N) & \end{array}$$

- $LN \xrightarrow{\widehat{E}} LN'$, where $N \xrightarrow{\widehat{E}} N'$.

  1. N is fully specified.

$$\begin{array}{ll}\mathcal{F}_{\widehat{E}}(LN) & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N) & >_{\text{i.h.}} \\ \mathcal{F}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N') & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(LN') & \end{array}$$

  2. N is not fully specified. Then, there are two cases:

     - N' is not fully specified.

$$\begin{array}{ll}\mathcal{F}_{\widehat{E}}(LN) & =_{\text{Lemma 9}} \\ \mathcal{E}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N) & >_{\text{i.h.}} \\ \mathcal{E}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N') & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(LN') & \end{array}$$

     - N' is fully specified.

$$\begin{array}{ll}\mathcal{F}_{\widehat{E}}(LN) & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N) & >_{\text{i.h.}} \\ \mathcal{F}_{\widehat{E}}(L) + \mathcal{E}_{\widehat{E}}(N') & =_{\text{Lemma 9}} \\ \mathcal{F}_{\widehat{E}}(LN') & \end{array}$$

Proof of Theorem 10 □

We can now state the confluence property for both reductions.

Theorem 11 (Confluence of $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$) *The relations $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$ are confluent.*

Proof of Theorem 11 (Confluence of $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$) First check that $\xrightarrow{E}$ and $\xrightarrow{\widehat{E}}$ are weakly confluent by inspecting all the critical pairs. Then the relation $\xrightarrow{E}$ (respectively, $\xrightarrow{\widehat{E}}$) is confluent as a consequence of Theorem 10 and Newman's lemma.

Proof of Theorem 11 □

4.3 Properties of Expansive Normal Forms

An important property of a one-step expansion $M \xrightarrow{\eta} M'$, $M \xrightarrow{\widehat{\eta}} M'$, 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 $\xrightarrow{\eta}$, $\xrightarrow{\widehat{\eta}}$, and $\xrightarrow{sp}$. This is seen clearly using a simple example.

Example 9 Let z be a variable of type $A_1 \times A_2$, where the A_i's are base types. Applying an sp-expansion to z, its E-normal form $\langle \pi_1(z), \pi_2(z) \rangle$ is obtained by inserting the variable z in the holes of the context $\langle \pi_1([\,]), \pi_2([\,]) \rangle$.

End of Example 9 □

In the general case, there are many positions of a term M where an expansion can be performed, so the characterization of its expansive normal form must take into account the *structure* of M, as suggested in Definition 14. Coming back to the previous Example 9 to define the E-normal form of the term $\langle z, z \rangle$, each occurrence of the variable z must be inserted in the holes of an expansor context, but the pair itself need not be. This is because each occurrence of z in $\langle z, z \rangle$ is reducible by $\xrightarrow{sp}$ but the pair $\langle z, z \rangle$ is not reducible at the root position.

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 η- or $\widehat{\eta}$-expanded and the term $\pi_2(z)$ can be sp-expanded. 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 (as suggested by the following definition) all of the information on possible future expansions that is fully contained in the *type* of the term being marked.

Definition 13 (Expansor Contexts) *Let C be one of the reduction relations E or $\widehat{E}$. Associate to every type D an* expansor context $\Delta^C_D[\,]$, *and to every pattern $P :: A$ a term $\widehat{P}$, both defined by mutual induction as follows:*

- *If P is a variable x, then $\widehat{P} = \Delta^C_A[P]$.*

[1] To differentiate with the expansor *terms* defined in [CK94].

- *If P is a pair $\langle P_1, P_2\rangle$, then* $\widehat{P} = \langle \widehat{P_1}, \widehat{P_2}\rangle$.
- *If D is an arrow type $D_1 \to D_2$, then* $\Delta^{\widehat{E}}_{D_1\to D_2}[] = \lambda P : D_1.\Delta^{\widehat{E}}_{D_2}[[]\widehat{P}]$, *where $P :: D_1$ and the variables of P are fresh.*
- *If D is an arrow type $D_1 \to D_2$, then* $\Delta^{E}_{D_1\to D_2}[] = \lambda x : D_1.\Delta^{E}_{D_2}[[]\Delta^{E}_{D_1}[x]]$, *where x is a fresh variable.*
- *If D is a product type $D_1 \times D_2$, then* $\Delta^{C}_{D_1\times D_2}[] = \langle \Delta^{C}_{D_1}[\pi_1[]], \Delta^{C}_{D_2}[\pi_2[]]\rangle$.
- *If D is a base type, then $\Delta^C_D[]$ is the empty context.*

One can write $\Delta^C_D?[M]$ to denote either $\Delta^C_D[M]$ or M. If M is typable, then one can check by induction on the structure of terms that $\Delta^C_A[M]$ is also typable, as follows.

Lemma 10 *If $\Gamma \triangleright M : A$, then $\Gamma \triangleright \Delta^C_A[M] : A$.*

The next step is to define the translation of a term M, which permits the insertion of each C-reducible subterm of M into the expansor context corresponding to its type.

Definition 14 (Translation) *Let C be one of the reduction relations E or $\widehat{E}$. For every term M of type A, its* translation $\mathcal{T}^C(M)$ *is defined as follows:*

$$\mathcal{T}^E(M) = \begin{cases} \mathcal{U}^E(M) & \textit{if } M = \langle P_1, P_2\rangle \textit{ or } M = \lambda x : B.N \\ \Delta^E_A[\mathcal{U}^E(M)] & \textit{otherwise} \end{cases}$$

$$\mathcal{T}^{\widehat{E}}(M) = \begin{cases} \mathcal{U}^{\widehat{E}}(M) & \textit{if } M = \langle P_1, P_2\rangle, \textit{ or } M = \lambda P : B.N \textit{ and } P :: B \\ \Delta^{\widehat{E}}_A[\mathcal{U}^{\widehat{E}}(M)] & \textit{otherwise} \end{cases}$$

and $\mathcal{U}^E(M)$ and $\mathcal{U}^{\widehat{E}}(M)$ are defined by induction on the structure of M as

$$\begin{array}{ll} \mathcal{U}^E(x) & = x \\ \mathcal{U}^E(\pi_i(M)) & = \pi_i(\mathcal{U}^E(M)) \\ \mathcal{U}^E(\lambda P : B.M) & = \lambda P : B.\mathcal{T}^E(M) \\ \mathcal{U}^E(\langle M, N\rangle) & = \langle \mathcal{T}^E(M), \mathcal{T}^E(N)\rangle \\ \mathcal{U}^E(MN) & = (\mathcal{U}^E(M)\mathcal{T}^E(N)) \end{array}$$

$$\begin{array}{ll}
\mathcal{U}^{\widehat{E}}(x) & = x \\
\mathcal{U}^{\widehat{E}}(\pi_i(M)) & = \pi_i(\mathcal{U}^{\widehat{E}}(M)) \\
\mathcal{U}^{\widehat{E}}(\lambda P : B.M) & = \lambda P : B.\mathcal{T}^{\widehat{E}}(M) \\
\mathcal{U}^{\widehat{E}}(\langle M, N\rangle) & = \langle \mathcal{T}^{\widehat{E}}(M), \mathcal{T}^{\widehat{E}}(N)\rangle \\
\mathcal{U}^{\widehat{E}}(MN) & = \begin{cases} (\mathcal{U}^{\widehat{E}}(M)\mathcal{T}^{\widehat{E}}(N)) & \textit{if } N \textit{ is fully specified} \\ (\mathcal{T}^{\widehat{E}}(M)\mathcal{T}^{\widehat{E}}(N)) & \textit{if } N \textit{ is not fully specified} \end{cases}
\end{array}$$

Example 10 Let C be one of the reduction relations E or $\widehat{E}$. Let v be a variable of type $(A \times (A' \to A'')) \to B$ and w be of type $D \times D'$, where A, A', A'', B, D, and D' are base types. Then

$$\begin{aligned}
\mathcal{T}^C(\langle v, w\rangle) = \mathcal{U}^C(\langle v, w\rangle) = \langle \mathcal{T}^C(v), \mathcal{T}^C(w)\rangle \\
= \langle \Delta^C_{(A\times(A'\to A''))\to B}[v], \Delta^C_{D\times D'}[w]\rangle
\end{aligned}$$

where

$$\begin{array}{ll}
\Delta^E_{D\times D'}[w] & = \Delta^{\widehat{E}}_{D\times D'}[w] = \langle \pi_1(w), \pi_2(w)\rangle \\
\Delta^E_{(A\times(A'\to A''))\to B}[v] & = \lambda x : A \times (A' \to A'').v\langle \pi_1(x), \lambda z : A'.\pi_2(x)z\rangle \\
\Delta^{\widehat{E}}_{(A\times(A'\to A''))\to B}[v] & = \lambda\langle x, y\rangle : A \times (A' \to A'').v\langle x, \lambda z : A'.yz\rangle
\end{array}$$

End of Example 10 □

It is easy to prove that the translation preserves types and characterizes E-normal forms.

Lemma 11 *For* $C \in \{E, \widehat{E}\}$*, if* $\Gamma \rhd M : A$*, then* $\Gamma \rhd \mathcal{T}^C(M) : A$ *and* $\Gamma \rhd \mathcal{U}^C(M) : A$*.*

Proof of Lemma 11 By induction on the structure of M.

Proof of Lemma 11 □

Any term that is only C-reducible at the root position is said to be in *C-quasinormal form*. As a consequence, every C-normal form is a C-quasinormal form. It turns out that for every term M, $\mathcal{T}^C(M)$ characterizes the C-normal form of M, while $\mathcal{U}^C(M)$ characterizes the C-quasinormal form of M.

Lemma 12 (Characterization) *Let C be one of the reduction relations E or $\widehat{E}$, and let $\Gamma \triangleright M : A$. Then $\mathcal{U}^C(M)$ is the C-quasinormal form of M and $\mathcal{T}^C(M)$ is the C-normal form of M.*

Proof of Lemma 12 (Characterization) By induction on the structure of M.

Proof of Lemma 12 $\square$

It is now possible to complete the proof of confluence of $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ by showing that for every one-step reduction $\xrightarrow{B}$ from M to N, there is a nonempty $\xrightarrow{B}$ reduction sequence from the C-normal form of M to the C-normal form of N.

Theorem 12 (Projection) *Let C be one of the reduction relations E and $\widehat{E}$. If $\Gamma \triangleright M : A$ and $M \xrightarrow{B} N$, then $\mathcal{T}^C(M) \xrightarrow{B}{}^{+} \mathcal{T}^C(N)$.*

Proof of Theorem 12 (Projection) By induction on the structure of M using the following facts:

1. $\Delta_A^C?[\mathcal{T}^C(M)[\overline{\mathcal{T}^C(L)}/\overline{x}]] \xrightarrow{\beta,\pi_i}{}^{*} \mathcal{T}^C(M[\overline{L}/\overline{x}])$
2. $\Delta_A^C[\mathcal{T}^C(M)] \xrightarrow{\beta,\pi_i}{}^{*} \mathcal{T}^C(M)$

Proof of Theorem 12 $\square$

From Lemma 12 and Theorem 12 we obtain the following corollary.

Corollary 2 *Let $C \in \{E, \widehat{E}\}$. If $\Gamma \triangleright M : A$ and $M \xrightarrow{B} N$, then $M^C \xrightarrow{B} {}^{+}N^C$.*

Theorem 13 (Confluence of $\xrightarrow{lwp}$ and $\xrightarrow{lw}$) *The relations $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ are* confluent.

Proof of Theorem 13 (Confluence of $\xrightarrow{lwp}$ and $\xrightarrow{lw}$) Using Theorems 6, 8, 10, and 11, and Corollary 2.

Proof of Theorem 13 $\square$

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

It is worth noticing that confluence and strong normalization of $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ can also be shown using the proof techniques in [Ges90, CP95, Cos96], which can be summarized by the following abstract theorem.

Theorem 14 (Modular Confluence and Strong Normalization) *Let* $\langle \mathcal{A}, \xrightarrow{R} \rangle$ *and* $\langle \mathcal{A}, \xrightarrow{S} \rangle$ *be two abstract reduction systems such that*

- $\xrightarrow{R}$ *and* $\xrightarrow{S}$ *are strongly normalizing,*
- $\xrightarrow{R}$ *and* $\xrightarrow{S}$ *are confluent,*
- $\xrightarrow{R}$ *preserves* S*-normal forms, and*
- *the following commutation holds:* $\forall M, N, L \in \mathcal{A}$, *if* $M \xrightarrow{R} L$ *and* $M \xrightarrow{S} N$, *then there is* $K \in \mathcal{A}$ *such that* $L \xrightarrow{S}{}^{*} K$ *and* $N \xrightarrow{R}{}^{+} K$.

Then $\xrightarrow{R} \cup \xrightarrow{S}$ *is confluent and strongly normalizing.*

If one takes R as B and S as E or $\widehat{E}$, confluence and strong normalization of $\xrightarrow{lwp}$ and $\xrightarrow{lw}$ can be derived using Theorems 8, 9, 10, and 11, and the following properties.

Property 1 (Preservation of E and $\widehat{E}$ Normal Forms) *B preserves E and $\widehat{E}$ normal forms.*

Property 2 (Commutation) *Whenever* $a \xrightarrow{B} c$ *and* $a \xrightarrow{E} b$ *(respectively,* $a \xrightarrow{\widehat{E}} b$*), there is* d *(respectively,* d'*) such that* $b \xrightarrow{B}{}^{+} d$ *(respectively,* $b \xrightarrow{B}{}^{+} d'$*) and* $c \xrightarrow{E}{}^{*} d$ *(respectively* $c \xrightarrow{\widehat{E}}{}^{*} d'$*).*

The preservation of E and $\widehat{E}$ normal forms can be shown by a more specific lemma.

Lemma 13 *If M is a C-quasinormal form (respectively, C-normal form) and $M \xrightarrow{B} M'$, then M' is also a C-quasinormal form (respectively, C-normal form).*

Proof of Lemma 13 By induction on the structure of M using the fact that C-quasinormal forms and C-normal forms are stable by substitutions of the form $\theta(x) = N$, where N is a C-normal form.

Proof of Lemma 13 □

The commutation property can be shown by inspecting all the critical pairs between B and E (respectively, B and $\widehat{E}$), and using some properties relating expansions and substitutions. However, checking local confluence in an expansionary calculus is not straightforward, because reductions (in particular expansion rules) are not stable by substitutions. The proof of the commutation property is omitted here: the reader is referred to [CK94] for more details about local confluence in a typed λ-calculus with expansion rules.

5 Adding Recursion to the LWPP Calculus

Here the focus is on adding fixpoint operators to our rewriting systems. A new constant $\mathit{fix}_A : (A \to A) \to A$ is assumed for each type A, with the reduction rule

$$\mathit{fix}_A \xrightarrow{\mathit{fix}} \lambda f.f(\mathit{fix}_A\ f) \tag{fix}$$

To simplify notations, the type suffix is dropped in the rest of this section.

It can be stated that $\xrightarrow{B\cup \mathit{fix}}{}^*$ is confluent, by the following theorem taken from [CK96].

Theorem 15 (Modularity of Confluence with *fix*) *Let L be any* left-linear *reduction system. If L is confluent, then so is L plus recursion.*

This result can be extended to $\xrightarrow{lw\cup fix}$ by the following theorem taken from [CK96]:

Theorem 16 (Modularity of Confluence with *fix* and Expansions) *Let L be any* left-linear *reduction system* containing *β and extended with expansion rules for η and surjective pairing. If L is confluent, then so is L plus recursion.*

To show that $\stackrel{lwp\cup fix}{\longrightarrow}$ is also confluent, it is sufficient to remark that the same proof provided in [CK96] for Theorem 16 also works if η is replaced by $\widehat{\eta}$. So we are able to draw the following conclusion.

Corollary 3 $\stackrel{lw\cup fix}{\longrightarrow}$ *and* $\stackrel{lwp\cup fix}{\longrightarrow}$ *are confluent reduction relations.*

6 Conclusion and Future Work

Two confluent reduction systems have been provided for the extensional typed λ-calculus with wildcard, layered, and product patterns equipped with fixpoints and product types.

It was shown that even if wildcard, layered, and product patterns allow more friendly specifications of function definitions (as Example 1 shows), they are redundant and can be eliminated. That is, for *any* function $\lambda P : A.M$, there is an equivalent function $\lambda z : A.M'$ modulo the equational theory $=_{\text{LWPP}}$.

It was also shown that only wildcard and layered patterns can be eliminated if one wants to keep product patterns. That is, for *any* function $\lambda P : A.M$, there is an equivalent function $\lambda P' : A.M'$ such that P' contains only variables and pairs. This fact is not very surprising, because it corresponds to the fact that contraction and weakening rules are redundant in intuitionist logic [Gal93], and simultaneously gives another possible reading: removing (at compile time) wildcard and layered patterns from λ-abstractions using this reduction system could lead to optimizations of programs that will be then interpreted by simpler computation rules.

The reduction system for the LWPP calculus without recursion can also be used as a decision method for equational provability, as a consequence of Theorems 2, 3, and 13. This is clearly illustrated by the examples in the paper.

A clean and formal proof of strong normalization of the expansive interpretation of η and sp has been provided. This proof has always been given in the literature by rather informal arguments.

This work clarifies the relation between extensionality and patterns, but it rests as future work to develop 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 types [CK94], combinations of algebraic rewriting systems with extensional λ-calculi [CK96], or extensional polymorphic λ-calculus [CK95].

7 Acknowledgments

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.

A LWPP and the Typed Pattern Calculus

The following result link the LWPP calculus to the typed pattern calculus (TPC).

Lemma 14 (Simulating LWPP in the TPC) *For every term M in the LWPP calculus, there is a translated term $\mathcal{S}(M)$ in the TPC such that $M \xrightarrow{B} M'$ implies $\mathcal{S}(M) \xrightarrow{\text{TPC}}{}^{+} \mathcal{S}(M')$.*

Proof of Lemma 14 (Simulating LWPP in the TPC) First define the translation $\mathcal{S}$ from the LWPP calculus into the TPC, then show the property by case analysis.

$$
\begin{array}{ll}
\mathcal{S}(x) & = x \\
\mathcal{S}(\langle M, N\rangle) & = \langle \mathcal{S}(M), \mathcal{S}(N)\rangle \\
\mathcal{S}(\lambda P : A.M) & = \lambda P : A.\mathcal{S}(M) \\
\mathcal{S}(\pi_i(M)) & = \textbf{let } \mathcal{S}(M) \textbf{ be } \langle x_1, x_2\rangle : A \times B \textbf{ in } x_i \\
\mathcal{S}(MN) & = \textbf{let } \mathcal{S}(M) \textbf{ be } \sharp z : A \to B \textbf{ in } (z \textbf{ of } \mathcal{S}(N) \textbf{ is } y \textbf{ in } y)
\end{array}
$$

where x_1, x_2, y, and z are fresh variables.

It is easy to prove that $\mathcal{S}(K)[\text{Match}(\mathcal{S}(L), P)] \xrightarrow{\text{TPC}}{}^{*} \mathcal{S}(K[\text{Match}(L, P)])$, so the lemma may then be proved by cases:

- $M = \pi_i(\langle N_1, N_2\rangle) \xrightarrow{\pi_i} N_i = M'$.

$$
\begin{array}{ll}
\mathcal{S}(\pi_i(\langle N_1, N_2\rangle)) & = \\
\textbf{let } \mathcal{S}(\langle N_1, N_2\rangle \textbf{ be } \langle x_1, x_2\rangle : A \times B \textbf{ in } x_i & = \\
\textbf{let } \langle \mathcal{S}(N_1), \mathcal{S}(N_2)\rangle \textbf{ be } \langle x_1, x_2\rangle : A \times B \textbf{ in } x_i & \xrightarrow{\text{TPC}} \\
\mathcal{S}(N_i) &
\end{array}
$$

- $M = (\lambda P : A.K)L \xrightarrow{\beta} K[\mathrm{Match}(L,P)] = M'$.

$$
\begin{array}{ll}
\mathcal{S}((\lambda P : A.K)L) & = \\
\textbf{let } \mathcal{S}(\lambda P : A.K) \textbf{ be } \sharp z : A \to B \textbf{ in } (z \textbf{ of } \mathcal{S}(L) \textbf{ is } y \textbf{ in } y) & = \\
\textbf{let } \lambda P : A.\mathcal{S}(K) \textbf{ be } \sharp z : A \to B \textbf{ in } (z \textbf{ of } \mathcal{S}(L) \textbf{ is } y \textbf{ in } y) & \xrightarrow{\mathrm{TPC}} \\
\lambda P : A.\mathcal{S}(K) \textbf{ of } \mathcal{S}(L) \textbf{ is } y \textbf{ in } y & \xrightarrow{\mathrm{TPC}} \\
\textbf{let } \mathcal{S}(K)[\mathrm{Match}(\mathcal{S}(L),P)] \textbf{ be } y \textbf{ in } y & \xrightarrow{\mathrm{TPC}} \\
\mathcal{S}(K)[\mathrm{Match}(\mathcal{S}(L),P)] & \xrightarrow{\mathrm{TPC}}{}^{*} \\
\mathcal{S}(K[\mathrm{Match}(L,P)]) &
\end{array}
$$

Proof of Lemma 14 □

References

[Aka93] Yohji Akama. On Mints' reductions for ccc-calculus. In M. Bezem and J. F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA)*, number 664 in Lecture Notes in Computer Science, pages 1–12, Berlin, Germany, March 1993. Springer-Verlag.

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, The Netherlands, revised edition, 1984.

[BTKP93] Val Breazu-Tannen, Delia Kesner, and Laurence Puel. A typed pattern calculus. In *Eighth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 262–274, Los Alamitos, CA, June 1993. IEEE Computer Society Press.

[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'Ecole Normale Supérieur, Ecole Normale Supérieure, 45, rue d'Ulm, 75230 Paris, France, 1986.

[CK94] Roberto Di Cosmo and Delia Kesner. Simulating expansions without expansions. *Mathematical Structures in Computer Science*, 4:315–362, 1994.

[CK95] Roberto Di Cosmo and Delia Kesner. Rewriting with extensional polymorphic λ-calculus (extended abstract). In Hans Kleine B¨uning, editor, *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL)*, number 1092 in Lecture Notes in Computer Science, pages 215–232, Berlin, Germany, September 1995. Springer-Verlag.

[CK96] Roberto Di Cosmo and Delia Kesner. Combining algebraic rewriting, extensional lambda calculi, and fixpoints. *Theoretical Computer Science*, 169(2):201–220, 1996.

[Cos96] Roberto Di Cosmo. The power of simple diagrams. In Harald Ganzinger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications (RTA)*, number 1103 in Lecture Notes in Computer Science, pages 200–214, Berlin, Germany, July 1996. Springer-Verlag.

[CP95] Roberto Di Cosmo and Adolfo Piperno. Expanding extensional polymorphism. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA)*, number 902 in Lecture Notes in Computer Science, pages 139–153, Berlin, Germany, April 1995. Springer-Verlag.

[Čub92] Djordje Čubrić. Embedding of a free Cartesian closed category into the category of sets. Available at the URL `ftp://triples.math.mcgill.ca/pub/cubric`, 1992. Also distributed on the `types` mailing list as "On Free CCC".

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North Holland, Amsterdam, The Netherlands, 1990.

[Dou93] Daniel Dougherty. Some lambda calculi with categorical sums and products. In Claude Kirchner, editor, *Proceedings of the Fifth International Conference on Rewriting Techniques and Applications (RTA)*, number 690 in Lecture Notes in Computer Science, pages 137–151. Springer-Verlag, June 1993.

[FL96] Roland Fetting and Bernd Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In Harald Ganzinger, editor, *Proceedings of the Seventh International Conference on Rewriting Techniques and Applications (RTA)*, number 1103 in Lecture Notes in Computer Science, pages 347–361, Berlin, Germany, July 1996. Springer-Verlag.

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

[Ges90] Alfons Geser. *Relative termination.* Dissertation, Fakultät für Mathematik und Informatik, Universität Passau, 1990. Also available as: Report 91-03, Ulmer Informatik-Berichte, Universität Ulm, 1991.

[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, September 1976.

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

[JG95] Colin Barry Jay and Neil Ghani. The virtues of eta-expansion. *Journal of Functional Programming*, 5(2):135–154, April 1995.

[Kes94] Delia Kesner. Reasoning about layered, wildcard and product patterns. In Giorgio Levi and Mario Rodríguez-Artelejo, editors, *Proceedings of the Fourth International Conference on Algebraic and Logic Programming (ALP)*, number 850 in Lecture Notes in Computer Science, pages 253–268, Berlin, Germany, 1994. Springer-Verlag.

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Center Tracts*. CWI, Amsterdam, The Netherlands, 1980.

[KPT96] Delia Kesner, Laurence Puel, and Val Tannen. A typed pattern calculus. *Information and Computation*, 124(1):32–61, January 1996.

[L+93] Xavier Leroy et al. The Caml Light system, release 0.6. Software and documentation distributed by anonymous FTP on `ftp.inria.fr`, 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. Technical Report SR-94-06, Department of Computer Science, University of the Saarland, 1989.

[PJ87] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[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. In J. E. Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium*, pages 235–307, Amsterdam, The Netherlands, 1971. North Holland.

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