An extensional operational and axiomatic semantics for type-inference with recursion and algebraic data types.

Roberto Di Cosmo

LIENS (CNRS) - DMI
Ecole Normale Supérieure
45, Rue d'Ulm
75005 Paris - France
E-mail: dicosmo@dmi.ens.fr

Abstract

In this paper, we provide an operational semantics of core-ML with recursion and algebraic data types which agrees precisely with the usual axiomatic semantics *in the presence of extensionality axioms*. This result has been missed for a long time due to the traditional use of contractive reduction rules for familiar extensional equalities like η and Surjective Pairing: these rules do not interact nicely with the implicit polimorphic typing discipline that is the base of many strongly typed functional programming languages, as even such a fundamental property as Subject Reduction does not hold in their presence.

We use here the more satisfactory approach to extensionality that uses expansion rules, and then we can give a simple translation of (extensional) Core-ML into (extensional) System F that preserves types *and* allows a proper simulation of the reductions of Core-ML. This provides an operational notion of reduction for Core-ML that takes into account the extensional equalities on the arrow and the product types.

But this is not all: we also show how we can add, using some simple but powerful lemmas from the theory of rewriting, algebraic data types and recursion preserving confluence (hence determinacy) of the system under very liberal conditions.

To the author's best knowledge, this is the first satisfactory treatment of a polymorphic type inference systems in the presence of extensionality.
KEYWORDS: type-inference, language design, rewriting, ML, semantics, algebraic data types

1 Introduction

It has been noticed by many people that it is difficult to add reductions corresponding to extensional equalities to a calculus based on type-inference, like the one associated to ML. What happens is that the usual presentation of η and Surjective Pairing not only breaks confluence when adding algebraic data types, but it also, and more fundamentally, breaks a fundamental property of the calculus, subject reduction (the fact that a reduct t' of a term t can get all the types that t could get). It is possible to check this fact in a short session with your favorite ML style functional programming language.

Take this simple program that builds a pair of identity functions, then decomposes the pair and builds it up again via projection and pairing.

Example 1.1

```
#let splitpair =
     let  join = let pair x = (x,x) in let id x = x in pair id
#in (fst join, snd join);;
Value splitpair is (<fun>,<fun>) : ('a -> 'a) * ('b -> 'b)
```

□

Its most general type is `('a -> 'a) * ('b -> 'b)` and it would reduce, if we allow SP contraction, to the following

Example 1.2

```
#let splitpair =
     let join  = let pair x = (x,x) in let id x = x in pair id
#in join;;
Value splitpair is (<fun>,<fun>) : ('a -> 'a) * ('a -> 'a)
```

□

Anyway, `('a -> 'a) * ('a -> 'a)` is less general than `('a -> 'a) *('b -> 'b)`: we lost in the reduction the possibility to instantiate the two components of the product type to different types.

For this reason, these rules, useful for reasoning on programs, have simply been dropped up to now.

This is particularly unfortunate: an axiomatic semantics of a functional programming language usually includes extensional equalities (as they are also valid w.r.t. observational equivalence, when observing base types on terminating programs), and one would like to provide an operational semantics (in the form of reduction rules), that agrees with it.

Now, by using the expansionary presentation of η and Surjective Pairing, subject reduction can be preserved, at the price of introducing reduction rules that depend on the type of terms. The question is then if it is possible to prove confluence (and eventually strong normalization) for this system. We can no longer simply argue that confluence of the untyped lambda-calculus plus subject reduction for pure ML give us the answer for free: indeed, in the presence of a contractive rule for Surjective Pairing the untyped calculus is not confluent [Klo80], while there seems to be no sound way of using expansion rules in an untyped context.

There has been some work (in a simpler explicitly typed framework), where some workarounds are presented to handle extensionality [HM90], later largely improved in [DCK94a] using expansive rules, but this paper is, to the author's best knowledge, the first full solution for type-inference systems: we provide a simple proof of normalization and confluence for the core language based on a natural interpretation of the extensional ML system into the extensional System F, that we proved to be strongly normalizing and confluent in [DCK95b], then add modularly algebraic data types preserving confluence and normalization, and finally fixpoints preserving confluence.

Let us start by introducing the core system, then we will proceed by presenting the translation into (extensional) polymorphic lambda calculus, and to add algebraic rewriting and fixpoints.

Definition 1.3 (core-ML) *The formal system for (core) ML is made of the untyped lambda terms t that can be assigned a type in the type-assignment system given in Table 1.*

Remark 1.4 *Notice that in the (LET) we allow generalization over an arbitrary subset of the type variables not bound in the term variable environment. This is to simplify the proofs, but one can of course recover the usual system by always taking the largest subset in the typing judgements.*

Notation 1.5 *In what follows, we will often use Gen(A) to refer to a type $\forall X_1 \ldots X_n.A$ where $\{X_1 \ldots X_n\} \subseteq FTV(A) - FTV(\Gamma)$, if Γ is clear from the context.*

(VAR) $\quad \Gamma \vdash x : A[\tau_i/X_i]$ if x:A = $\forall X_1 \ldots X_n.\tau$ is in Γ and the τ_i are monotypes

$$(ABS) \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

$$(APP) \quad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B}$$

$$(PAIR) \quad \frac{\Gamma \vdash M : A_1 \quad \Gamma \vdash N : A_2}{\Gamma \vdash < M, N >: A_1 \times A_2}$$

$$(PROJ) \quad \frac{\Gamma \vdash M : A_1 \times A_2}{\Gamma \vdash \mathrm{p}_i M : A_i}$$

$$(LET) \quad \frac{\Gamma \vdash N : A \quad \Gamma, x : \forall X_1 \ldots X_n.A \vdash M : B}{\Gamma \vdash let \;\; x \;\; = \;\; N \;\; in \;\; M : B} \quad \text{where } \{X_1 \ldots X_n\} \subseteq FTV(A) - FTV(\Gamma)$$

Table 1: Type inference rules for an ML-like functional language.

1.1 Substitution of types and terms in derivations

We prove here some basic properties of the type assignment rules that are crucial to the proof of the subject reduction theorem, but also show how whenever a term can be assigned a type, then it can be assigned also all instances of that type.

Lemma 1.6 (Substitution of types in derivations) *Let $\Gamma \vdash M : A$ be a derivation and let X be any type variable . Then it is possible to build a derivation $\Gamma[\tau/X] \vdash M : A[\tau/X]$, for τ any given monotype.*

Proof. By induction on the derivation of $\Gamma \vdash M : A$. □

Lemma 1.7 (Minimal Environments) *If $\Gamma, x : B \vdash M : A$, and $x \notin FV(M)$, then also $\Gamma \vdash M : A$.*

Proof. By a simple induction of the derivation of $\Gamma, x : B \vdash M : A$. □

Lemma 1.8 (Environments can be extended) *If $\Gamma \vdash M : A$, then also $\Gamma \cup \Gamma' \vdash M : A$, if $\Gamma \cup \Gamma'$ is a well formed environment.*

Proof. By induction on the structure of the derivation. □

Lemma 1.9 (Substitution of terms in derivations) *Let $\Gamma, x : Gen(A) \vdash M : B$ and $\Gamma \vdash N : A$ be derivations, where Gen(A) is a generalization of A w.r.t. some of the type variables not free in Γ. Then it is possible to build a derivation $\Gamma \vdash M[N/x] : B$.*

Proof. By induction on the derivation of $\Gamma \vdash M : A$. □

2 Normalization and confluence for extensional ML

Here we present a notion of reduction on ML that we will prove confluent and strongly normalizing. Notice that, since we work in a type-assignment framework, reductions are relativized by a basis Γ where the types of the free term variables are declared.

Definition 2.1 *(Notion of reduction for ML)*

beta-eta:

(β) $\Gamma \vdash (\lambda x.M)N \overset{ML}{\longrightarrow} M[x := N]$, *if N is free for x in M*

(β) $\Gamma \vdash let x = N in M \overset{ML}{\longrightarrow} M[x := N]$, *if N is free for x in M*

(η_{exp}) $\Gamma \vdash M \overset{ML}{\longrightarrow} \lambda x.(Mx) : A \to B$, *if* $\Gamma \vdash M : A \to B$

for $x \notin FV(M)$, and M not a λ-abstraction

projections and surjective pairing:

(π) *if* $\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2$, $\Gamma \vdash p_i(\langle M_1, M_2 \rangle) \overset{ML}{\longrightarrow} M_i : A_i$

(SP_{exp}) $\Gamma \vdash M \overset{ML}{\longrightarrow} \langle p_1(M), p_2(M) \rangle : A \times B$,

if $\Gamma \vdash M : A \times B$, *for M not a pair*

These basic reductions are turned into a reduction relation by context closure, but with the condition that η expansion of applied terms and SP expansion of projected terms are forbidden, as in done for explicitly typed calculi [Aka93, Dou93, DCK94b, Cub92, JG92].

For our reduction system, it is possible to show the property that fails when using contraction rules for the extensional equalities.

Proposition 2.2 (Subject Reduction)

Given a derivation $\Gamma \vdash M : A$ and a reduction M$\overset{ML}{\longrightarrow}$M', one can find a derivation $\Gamma \vdash M' : A$

Proof. By induction on the derivation of $\Gamma \vdash M : A$, and by cases on the reduction.
We only consider here the case of a root expansions of a term M, for which one can always build the needed derivation as follows:

$$\dfrac{\dfrac{\Gamma, z : A \vdash M : A \to B \quad \Gamma, z : A \vdash z : A}{\Gamma, z : A \vdash Mz : B}}{\Gamma \vdash \lambda z.Mz : A \to B} \qquad \dfrac{\dfrac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathrm{p}_1 M : A} \quad \dfrac{\Gamma \vdash M : A \times B}{\Gamma \vdash \mathrm{p}_2 M : B}}{\Gamma \vdash \langle \mathrm{p}_1 M, \mathrm{p}_2 M \rangle : A \times B}$$

It suffices to notice that we can assume $z \notin \Gamma$, as it is a bound variable of $\lambda z.Mz$ that does not appear in M, so one can build the derivations $\Gamma, z : A \vdash M : A \to B$ and $\Gamma, z : A \vdash z : A$ by environment extensions (lemma 1.8).

It is interesting to notice here that if we were to deal with the contraction rules, such a simple argument would definitely not hold. Indeed, if $\Gamma \vdash \lambda z.Mz : A \to B$, it is not obvious that $\Gamma \vdash M : A \to B$, while if $\Gamma \vdash \langle \mathrm{p}_1 M, \mathrm{p}_2 M \rangle : A \times B$, then it is simply not true, as we remarked before, that $\Gamma \vdash M : A \times B$.
□

2.1 Mapping ML into system F

The translation e°_{Γ} of a term e of ML is given by induction on the derivation $\Gamma \vdash e : A$ of a typing in ML (here we make the environment Γ explicit only when necessary), so we actually translate derivations rather than simple terms.

- $x^{\circ}_{\Gamma} = x[\tau_1] \ldots [\tau_n]$ if $\Gamma \vdash x : C$ where $x : \forall X_1 \ldots X_n.\tau \in \Gamma$, and $C = \tau[\overrightarrow{\tau_i}/\overrightarrow{X}]$
- $let\ x\ =\ N\ in\ M^{\circ}_{\Gamma} = (\lambda x : Gen(A).M^{\circ}_{\Gamma, x:Gen(A)})(\lambda \overrightarrow{X}.N^{\circ}_{\Gamma})$ if $\Gamma \vdash let\ x\ =\ N\ in\ M : B$ is derived via the (LET) rule from $\Gamma, Gen(A) \vdash M : B$ and $\Gamma \vdash N : A$, with the abstraction of the type variables $\overrightarrow{X}$ of A in Gen(A).

- $(MN)^\circ_\Gamma = (M^\circ_\Gamma N^\circ_\Gamma)$
- $\langle M_1, M_2\rangle^\circ_\Gamma = \langle M_1{}^\circ_\Gamma, M_2{}^\circ_\Gamma\rangle$
- $\mathrm{p}_i M^\circ_\Gamma = \mathrm{p}_i M^\circ_\Gamma$
- $\lambda x.M^\circ_\Gamma = \lambda x : A.M^\circ_{\Gamma,x:A}$ if $\Gamma\vdash\lambda x.M : A \to B$

One can show by induction on the derivation of $\Gamma\vdash e : A$ of a typing in ML that

Proposition 2.3 (Properties of the translation I)

1. *for any derivation $\Gamma\vdash e : A$, its translation e°_Γ is a well typed term of F with the same type in the same environment Γ.*
2. *for any derivation $\Gamma\vdash e : A$, with e in normal form, its translation e°_Γ is a term of F in normal form, with no type abstractions and such that erasure(e°_Γ)=e.*

Proof. By cases on the definition of the translation □

Lemma 2.4 *If $\Gamma\vdash M : A_1 \to A_2$, and $x \notin FV(M)$, then we can find a derivation of $\Gamma\vdash\lambda x.Mx : A_1 \to A_2$ s.t. $\lambda x.Mx^\circ_\Gamma$ is $\lambda x : A_1.((M^\circ_\Gamma)x)$.*

Proof. □

Proposition 2.5 (Properties of the translation II: Simulation)
If $\Gamma\vdash e : A$ and e reduces in one step to e', then there exists a derivation $\Gamma\vdash e' : A$ s.t. e°_Γ reduces in at least one step to e'°_Γ,

Proof. By induction on the definition of the translation, and by cases on e. □

Lemma 2.6 (Substitution vs translation, simple case) *If $\Gamma, x : A\vdash M : B$ and $\Gamma\vdash N : A$, where $x \notin FV(N)$, then there is a derivation $\Gamma\vdash M[N/x] : B$ s.t. $M^\circ_{\Gamma,x:A}[N^\circ_\Gamma/x] =_{Fexp} M[N/x]^\circ_\Gamma$.*

Proof. By cases on the definition of the translation of M. □

Lemma 2.7 (Substitution vs translation) *If $\Gamma, x : Gen(A)\vdash M : B$ and $\Gamma\vdash N : A$, then there is a derivation $\Gamma\vdash M[N/x] : B$ s. t. $M^\circ_{\Gamma,x:Gen(A)}[\lambda\overrightarrow{X}.N^\circ_\Gamma/x] \xrightarrow{Fexp} M[N/x]^\circ_\Gamma$, with $\overrightarrow{X}$ the type variables of A abstracted in Gen(A).*

Proof. By cases on the definition of the translation of M. □

This is enough to establish the following results about (core) ML:

Lemma 2.8 (Strong normalization of $\xrightarrow{ML}$) *The reduction relation $\xrightarrow{ML}$ for ML is strongly normalizing.*

Proof. An infinite reduction sequence leaving a term e in ML would give raise to an infinite reduction sequence in F leaving e°_Γ, which is impossible, as this last system is strongly normalizing. □

Lemma 2.9 (Confluence of $\xrightarrow{ML}$) *The reduction relation $\xrightarrow{ML}$ for ML is confluent.*

Proof. Let e be a term s.t. $e_1 \,{}^*\!\!\Longleftarrow e \Longrightarrow^* e_2$. Since $\Longrightarrow$ is strongly normalizing, we can reduce the terms e_i to their normal forms $\overline{e_i}$. Then we have $\overline{e_1} \,{}^*\!\!\Longleftarrow e \Longrightarrow^* \overline{e_2}$, and by proposition 2.5 $\overline{e_1}^\circ_\Gamma \,{}^+\!\!\Longleftarrow e^\circ_\Gamma \Longrightarrow^+ \overline{e_2}^\circ_\Gamma$ in F with expansions. As the translation of an ML normal form is a normal form in F, and F with expansion is confluent, we get that $\overline{e_1}^\circ_\Gamma = e_3 = \overline{e_2}^\circ_\Gamma$. Now, $\overline{e_1} = erasure(\overline{e_1}^\circ_\Gamma) = erasure(e_3) = erasure(\overline{e_2}^\circ_\Gamma) = \overline{e_2}$ allows us to conclude. The following figure shows the reduction diagram:

$$\begin{tikzcd}
 & e \arrow[dl, "*"'] \arrow[dr, "*"] \arrow[dd, dotted, no head] & \\
e_1 \arrow[d, "*"] & & e_2 \arrow[d, "*"] \\
\overline{e_1} \arrow[d, equal] & e^\circ_\Gamma \arrow[dl, "*"] \arrow[dr, "*"] & \overline{e_2} \arrow[d, equal] \\
\overline{e_1}^\circ_\Gamma & = & \overline{e_2}^\circ_\Gamma
\end{tikzcd}$$

□

3 Adding algebraic data structures

Now that our extensional core language for ML is set up, and proven confluent and strogly normalizing, it is time to focus our attention to the addition of algebraic data types. These can be added by means of *canonical* (that is, confluent and strongly normalizing) algebraic rewriting system. Here we want to show, in the spirit of [BTG94, DCK94a, JO94], that their addition to the core language *preserves* strong normalization and confluence.

There are at least three ways to prove this modularity result:

- Take directly one modularity proof for a type-inference system like the one given in [Bar90] and extend it to expansive extensional type-inference systems, using the same lemma from [DCP95].
- Redo the proof of modularity via translation as in [DCK94b].
- Take the modularity result shown in [BTG94] for non-extensional System F and algebraic rewriting system, extend it to expansive extensional System F using a lemma from [DCP95], and lift it to core-ML along our translation into System F.

This last strategy yields a particularly simple proof, even if, having also to prove the result for System F plus A, it does take some more space than just reusing the results in [Bar90]. Due to space limitations we decided to go for the first proof strategy, even if the last one is really what we like better.

Lemma 3.1 (Lemma of [DCP95]) *Let $\langle \mathcal{A}, \xrightarrow{R}, \xrightarrow{S} \rangle$ be an Abstract Reduction System, where R-reduction is strongly normalizing. Let the following commutation hold*

$$\forall a, b, c, d \in \mathcal{A} \quad \begin{tikzcd}
a \arrow[r, "R"] \arrow[d, "S"] & c \arrow[d, dashed, two heads, "S"] \\
b \arrow[r, dashed, two heads, "R", "+"'] & d
\end{tikzcd}$$

then R^+ and S^* commute.

Theorem 3.2 (Modularity of confluence and strong normalization) *If A is a confluent and strongly normalizing algebraic rewriting system, then Core-ML+A is confluent.*

Proof. We know from [Bar90], that the combination of Core-ML *without* extensional rules with A is also strongly normalizing and confluent. It is then easy to check the following diagram:

$$\begin{tikzcd}
a \arrow[r, "core-ML"] \arrow[d, "exp"] & c \arrow[d, dashed, two heads, "exp"] \\
b \arrow[r, dashed, two heads, "core-ML", "+"'] & d
\end{tikzcd}$$

Using lemma 3.1, since core-ML is strongly normalizing *and* preserves expansive normal forms, while expansion rules alone are confluent and strongly normalizing (for this see [Min77, Kes93, DCK94a]), we obtain (n.f. exp stands for reduction to normal form w.r.t expansion rules alone):

$$\begin{tikzcd}
a \arrow[r, "core-ML"] \arrow[d, "n.f."', "exp"] & c \arrow[d, "n.f."', "exp"] \\
b \arrow[r, dashed, two heads, "core-ML", "+"'] & d
\end{tikzcd}$$

Also, by a simple induction on the structure of a, one gets

$$\begin{tikzcd}
a \arrow[r, "\mathcal{A}"] \arrow[d, "n.f."', "exp"] & c \arrow[d, "n.f."', "exp"] \\
b \arrow[r, dashed, two heads, "\mathcal{A}", "+"'] & d
\end{tikzcd}$$

Putting all together, we arrive at :

$$\begin{tikzcd}
a \arrow[r, "core-ML+\mathcal{A}"] \arrow[d, "n.f."', "exp"] & c \arrow[d, "n.f."', "exp"] \\
b \arrow[r, dashed, two heads, "core-ML+\mathcal{A}", "+"'] & d
\end{tikzcd}$$

This diagram allows to deduce:

- strong normalization for the full system (as any infinite reduction sequence in the full system can be turned via the diagram into an infinite reduction system for core-ML+A, which would be an absurd)
- confluence of the full system: core-ML+A is confluent, and η, SP alone are confluent and strongly normalizing, so the previous commutation property can be used to close any divergent diagram proceeding quite like in lemma 2.9, using expansive normal forms as the "translation"

$$\begin{tikzcd}
 & & e \arrow[dll, "*"] \arrow[drr, "*"] \arrow[dd, dotted, no head] & & \\
e_1 \arrow[d, "*"] & & & & e_2 \arrow[d, "*"] \\
exp-n.f.(e_1) \arrow[d, equal] & & exp-n.f.(e) \arrow[dll, "ML+\mathcal{A}"', "*"] \arrow[drr, "ML+\mathcal{A}", "*"'] & & exp-n.f.(e_2) \arrow[d, equal] \\
exp-n.f.(e_1) \arrow[drr, "*", "ML+\mathcal{A}"'] & & & & exp-n.f.(e_2) \arrow[dll, "*"', "ML+\mathcal{A}"] \\
 & & e' & &
\end{tikzcd}$$

The lower diagram is closed using the confluence of $ML + \mathcal{A}$.

□

4 Adding recursion

We have proved confluence and strong normalization for our core ML language, and even if we have added algebraic data types, we do not have recursion neither at the level of types nor at the level of terms, so we would like to extend now our results to a more realistic language, namely, by adding a fixpoint operator.

4.1 Recursive terms

Let us focus first on adding a fixpoint operator fix for terms, with the reduction rule

$$(fix) \qquad fix\ M \xrightarrow{fix} M(fix\ M)$$

Usually, to prove that a reduction relation with such a fixpoint operator is confluent, one considers an auxiliary reduction relation with *bounded* fixpoint operators fix^n with the more strict reduction rule

$$(fix^n) \qquad fix^n\ M \quad \xrightarrow{fix^n} \quad M(fix^{n-1}\ M) \quad n > 0$$

This trick essentially puts a bound on the depth of any recursive call, so the reduction relation with such a rule is usually still strongly normalizing; if we can show that local confluence also still holds, then by Newman's Lemma we have confluence of this auxiliary reduction relation, and then it is possible to derive the confluence of the reduction relation with unbounded recursion by means of an easy simulation trick essentially due to Lévy (see [Lév76]):

Remark 4.1 *If $M \xrightarrow{fix^n} N$, then $|M| \xrightarrow{fix} |N|$, where $|M|$ is obtained from M by removing all the indices from the fix terms.*

Lemma 4.2 *For any reduction sequence $M_0 \xrightarrow{fix} M_1 \xrightarrow{fix} \dots \xrightarrow{fix} M_n$, there exists an indexed computation $N_0 \xrightarrow{fix^n} N_1 \xrightarrow{fix^n} \dots \xrightarrow{fix^n} N_n$ such that $|N_i| = M_i$, for $i = 0 \dots n$.*

Proof. Index all the fix constructors in M_0 by a number $n + k$, with $k \geq 0$. $\square$

To fully follow this approach, one is left to prove two things:

1. the auxiliary reduction relation is strongly normalizing
2. the auxiliary reduction relation is locally confluent

This is not only combersome and repetitive (as it is usually a rewriting of an existing proof for the language without bounded fixpoints), but is also not necesary, as has been shown by Delia Kesner and the author in [DCK94a, DCK95a]: indeed, for *any* left-linear rewriting system, the addition of this fixpoint operator preserves confluence (but, obviously, not normalization).

Theorem 4.3 (Algebraic left-linear core-ML plus fixpoints is confluent) *If A is a canonical left-linear algebraic rewriting system, then core-ML + A + fix is confluent.*

Proof. A direct consequence of theorem 4.11 of [DCK94a][1]. The main idea of that theorem is that one can translate the language with bounded `fix` into the language without bounded `fix`, and then redo Levy's trick in a completely generic form, that reduces confluence in the presence of fixpoints to left-linearity and confluence without fixpoints (which is our case here). We refere the interested reader to [DCK94a, DCK95a] for the details of the proof. $\square$

Before concluding, let us remark that left-linearity is indeed required.
The reason is that if there is some rule (like the contractive version of Surjective Pairing) where some metavariable appears more than once, it is easy to build counterexamples like the following one to Lévy's trick:

Example 4.4

$$\begin{array}{ll} \multicolumn{2}{l}{(\lambda p.(\langle \mathrm{p}_1 p, \mathrm{p}_2 \langle \mathrm{p}_1 p, \mathrm{p}_2 p\rangle\rangle))(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle))} \\ \longrightarrow & \langle \mathrm{p}_1(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle)), \mathrm{p}_2\langle \mathrm{p}_1(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle)), \mathrm{p}_2(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle))\rangle\rangle \\ \twoheadrightarrow & \langle \mathrm{p}_1\langle \mathrm{p}_1(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle)), \mathrm{p}_2(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle))\rangle, \\ & \mathrm{p}_2\langle \mathrm{p}_1(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle)), \mathrm{p}_2(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle))\rangle\rangle \\ \xrightarrow{SP} & \langle \mathrm{p}_1(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle)), \mathrm{p}_2(fix(\lambda x.\langle \mathrm{p}_1 x, \mathrm{p}_2 x\rangle))\rangle \end{array}$$

$\square$

[1] Or, better theorem 4.9 of [DCK95a].

Whatever index n we associate to the original fix operator, there is no way to simulate this reduction in the labeled calculus, as the occurrences of fix in the last redex will have labels differing by 1.

5 Conclusion

Using the expansionary presentation of η and Surjective Pairing, we showed that subject reduction in the type-inference framework can be preserved, at the price of introducing reduction rules that depend on the type of terms.

We then provided a natural interpretation of the extensional ML system into the extensional System F, that allows to give a simple proof of normalization and confluence for the core language. We also showed how algebraic rewriting can be very easily handled with expansive extensional rules (while it is quite incompatible with contractive extensional rules) preserving confluence and normalization, and finally we could add fixpoints to the whole system preserving confluence (and this preservation is now a simple corollary of a general lemma).

This set of results allows to fully reconcile the axiomatic and operational semantics of a realistic type-inference language like the one presented here. This is one of the main achievement of the paper, but perhaps another interesting contribution is, indeed, the very simplicity and clarity of the proof techniques used here: this should set the basis for a faster achievement of important analogous results for more complex languages.

References

[Aka93] Yohji Akama. On Mints' reductions for ccc-Calculus. In *Typed Lambda Calculus and Applications*, number 664 in LNCS, pages 1–12. Springer Verlag, 1993.

[Bar90] Franco Barbanera. Combining term rewriting and type assignment systems. *Int. Journal of Found. of Comp. Science*, 1:165–184, 1990.

[BTG94] Val Breazu-Tannen and Jean Gallier. Polymorphic rewiting preserves algebraic confluence. *Information and Computation*, 1994.

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

[DCK94a] Roberto Di Cosmo and Delia Kesner. Combining first order algebraic rewriting systems, recursion and extensional lambda calculi. In Serge Abiteboul and Eli Shamir, editors, *Intern. Conf. on Automata, Languages and Programming (ICALP)*, volume 820 of *Lecture Notes in Computer Science*, pages 462–472. Springer-Verlag, July 1994.

[DCK94b] Roberto Di Cosmo and Delia Kesner. Simulating expansions without expansions. *Mathematical Structures in Computer Science*, 4:1–48, 1994. A preliminary version is available as Technical Report LIENS-93-11/INRIA 1911.

[DCK95a] Roberto Di Cosmo and Delia Kesner. Combining algebraic rewriting, extensional lambda calculi and fixpoints. *Theoretical Computer Science*, 1995. Submitted.

[DCK95b] Roberto Di Cosmo and Delia Kesner. Rewriting with polymorphic extensional λ-calculus. In *CSL'95 (extended abstract)*, 1995. Full version submitted.

[DCP95] Roberto Di Cosmo and Adolfo Piperno. Expanding extensional polymorphism. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 139–153, April 1995.

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

[HM90] Brian Howard and John Mitchell. Operational and axiomatic semantics of pcf. In *Proceedings of the LISP and Functional Programming Conference*, pages 298–306. ACM, 1990.

[JG92] Colin Barry Jay and Neil Ghani. The Virtues of Eta-expansion. Technical Report ECS-LFCS-92-243, LFCS, 1992. University of Edimburgh, to appear inJournal of Functional Programming.

[JO91] Jean-Pierre Jouannaud and Mitsuhiro Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 350–361, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.

[JO94] Jean-Pierre Jouannaud and Mitsuhiro Okada. Executable higher-order algebraic specification languages. Draft (Extended Version of [JO91]), 1994.

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

[Klo80] Jan Willem Klop. Combinatory reduction systems. *Mathematical Center Tracts*, 27, 1980.

[Lév76] Jean-Jaques Lévy. An algebraic interpretation of the $\lambda\beta\kappa$-calculus and a labelled λ-calculus. *Theoretical Computer Science*, 2:97–114, 1976.

[Min77] Gregory Mints. Closed categories and the theory of proofs. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V.A. Steklova AN SSSR*, 68:83–114, 1977.