Exceptions in the rewriting calculus

Germain Faure[1] and Claude Kirchner[2]

[1] ENS Lyon & LORIA
46, allée d'Italie, 69364 Lyon Cedex 07, France
Germain.Faure@ens-lyon.fr
[2] LORIA & INRIA
615 rue du jardin botanique, 54602 Villers-lès-Nancy Cedex, France
Claude.Kirchner@loria.fr
www.loria.fr/~ckirchne

Abstract. In the context of the rewriting calculus, we introduce and study an exception mechanism that allows us to express in a simple way rewriting strategies and that is therefore also useful for expressing theorem proving tactics. This gives us the ability to simply express the semantics of the `first` tactical and to describe in full details the expression of conditional rewriting.

1 Introduction

The rewriting calculus makes all the basic ingredients of rewriting explicit objects, in particular the notions of rule formation i.e. *abstraction*, rule *application* and *result*. The original design of the calculus [CK01,Cir00] has good properties and encompasses lambda calculus as well as first-order rewriting in a uniform setting. Thanks to its matching power, *i.e.* the ability to discriminate directly from pattern *and* to the first class status of rewrite rules, it also has the capability to represent in a simple and natural way object calculi [CKL01a,DK00]. Thanks to its abstraction mechanism (the rule formation), it allows us to design extremely powerful type systems generalizing the lambda-cube [CKL01b].

One of the basic but already elaborated goal we want to reach using the ρ-calculus, is to use its matching-ability to naturally and simply express reduction and normalization strategies. Indeed, the first simple question to answer is: does it exist a ρ-term representing a given derivation in a rewrite theory? And the answer has been shown to be positive [CK01], leading in particular to the use of ρ-terms as rewrite derivations [Ngu01] proof terms. The next step consists to go from rewrite derivations to rewrite strategies, and in particular normalization ones. Therefore we want to answer the question: *given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u if u normalizes to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to (a set containing) the term v?*

Since the ρ-calculus embeds the λ-calculus, any computable function as the normalization one is expressible in this formalism. What our interest consist of, is to make use of the matching power and the non-determinism of the ρ-calculus to bring an increased ease in the expression of such functions together with their

expression in a uniform formalism combining standard rewrite techniques and high-order behaviors.

In seeking to use the specificities of the calculus to solve the previous questions, we reached several difficulties, the main one being the ambivalent use of the empty set.

For example, when applying on the term b the rewrite rule $a \to \emptyset$ that rewrites the constant a into the emptyset, the evaluation rule of the calculus returns $\emptyset$ *because matching against* b *fails*. This is denoted $[a \to \emptyset](b) \mapsto\!\!\!\!\to_{\rho} \emptyset$. But it is also possible to explicitly rewrite an object into the empty set like in $[a \to \emptyset](a) \mapsto\!\!\!\!\to_{\rho}^{*} \emptyset$, and in this case the result is also the empty set *because the rewrite rule explicitly introduces it*.

It becomes then clear that one should avoid the ambivalent use of the empty set and introduce an explicit distinction between failure and empty set of result. This leads us to the results presented in this paper where, by making this distinction, we add to the matching power of the rewriting calculus a *catching power*. This has for consequence to allow us to describe in a rewriting style exception mechanisms, and therefore to express easily convenient and elaborated strategies needed when evaluating or proving. Because of his fundamental role, we mainly focus on the `first` strategy combinator used in ELAN [BKK+98] that is also called IF THEN in LCF like logical frameworks.

So, we introduce a new constant $\perp$, and an exception handler operator exn to allow for an exception mechanism. In this new version of the ρ-calculus, like in most exception mechanisms, an exception can be either raised the user ($\perp$ is put explicitly in the initial term like for example in $[x \to \perp](a)$) or can be caused by an "run time error" (for example a rule application failure like in $[f(x,y) \to x](g(a,b))$). Either an exception can be *uncaught* ($\perp$ goes all over the term and the term is reduced to $\{\perp\}$ when using a strict evaluation strategy) or it can be stopped by exn and next *caught* thanks to matching. Then the calculus permits us to:
— catch the rule application failure (and therefore to be able to detect that a term is in normal form) like in $\big[exn(\perp) \to c\big]\big(exn\big([f(x,y) \to x](g(a,b))\big)\big)$
— ignore a raised exception like for example in the term $[x \to c](exn(M))$
— switch the evaluation mechanism according to a possible failure in the evaluation of a term (think again to the application to normal terms), using a term like $\big[\texttt{first}\big(exn(\perp) \to P_{failure}, x \to P_{normal}\big)\big]\big(exn(M)\big)$.

Our contributions are therefore the construction, study and use of a rewriting calculus having an explicit exception mechanism. In the next section, we motivate and build the calculus. In section 3, we prove it to be confluent when considering a call by value strategy. We finally show in section 4 that elaborated rewriting strategies can be described and evaluated, in particular we show that the `first` strategy combinator can be naturally expressed.

2 The ρ_ε-calculus

We introduce here a new ρ-calculus: the ρ_ε-calculus. After the construction of the $\rho^{1^{st}}$-calculus (the ρ-calculus doped with the `first` operator as described in [CK01]), it is quite natural to ask if the `first` operator can be simply expressed in the ρ-calculus. To understand the problem, we ask the question: instead of enriching the ρ-calculus with the `first` operator and its specific evaluation rules, what atomic ingredients must we add to the ρ-calculus to express the `first`? The first problem is to obtain a non-ambivalent meaning for the empty set, *i.e.* we must distinguish the rule application failure from the empty set of terms. The second problem is to obtain both the strictness and the rule application failure test.

2.1 Can the `first` be simply expressed in the ρ-calculus?

Since the ρ-calculus embeds the λ-calculus, any computable function like the `first` can be encoded in it. Using the "deep" and "shallow" terminology promoted in [BGG+92], what we are looking for is to *simply express* the `first` by avoiding the use of a deep encoding that do not fully use the matching capability of the ρ-calculus . What we call in this paper a *simple* encoding is a shallow encoding using in a natural way the matching capabilities of the framework.

The original ρ-calculus has been built in such a way that the empty set represents both an empty set of terms and the result of the failure of a rule application. For example, using the notations from [CK01] where $[\_](\_)$ is the application operator of a ρ-term (typically $l \to r$) on another one t, we have

$$[a \to \emptyset](a) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow^{*}_{\rho} \emptyset,$$

where $\emptyset$ represents an empty set of terms and

$$[a \to \emptyset](b) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow_{\rho} \emptyset,$$

where $\emptyset$ encodes the failure in the matching between a and b. So, in this version of the ρ-calculus, we are not able to determine if the result of a rule application is a failure or not. Nevertheless, this test is necessary if we want to define the `first` in the ρ-calculus, since the `first` is defined by:

$$\begin{aligned} First' \ & [\texttt{first}(s_1, \ldots, s_n)](t) \rightsquigarrow \{u_k \downarrow\} \\ & \qquad \text{if } [s_i](t) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow^{*}_{\rho} \emptyset, 1 \leqslant i \leqslant k-1 \\ & \qquad\quad [s_k](t) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow^{*}_{\rho} u_k \downarrow \neq \emptyset,\ \mathcal{FV}ar(u_k\downarrow) = \emptyset \end{aligned}$$

$$\begin{aligned} First'' \ & [\texttt{first}(s_1, \ldots, s_n)](t) \rightsquigarrow \emptyset \\ & \qquad \text{if } [s_i](t) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow^{*}_{\rho} \emptyset, 1 \leqslant i \leqslant n \end{aligned}$$

where $u_k\downarrow$ denotes the normal form of u_k for the evaluation rules of the ρ-calculus (denoted $\mapsto\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow^{*}_{\rho}$) and $\mathcal{FV}ar(t)$ denotes the free variables of t.

The side condition in $First'$ indicates that u_k must not contain any free variable and must be in normal form seems to be restrictive at first sight, but it is here to include conditions for the results so as to allow a good definition of the `first` and to get confluence.

Despite serious efforts to find it, the `first` seems not to be simply expressible in the ρ-calculus, and even if it were, there is a serious drawback in not distinguishing the empty set from the rule application failure. Indeed, $\big[\texttt{first}\big(a \to \emptyset, b \to c\big)\big]\big(a\big) \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \emptyset$ and $\big[\texttt{first}\big(a \to \emptyset, b \to c\big)\big]\big(c\big) \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \emptyset$ for two very different reasons. So, this provides us with a strong motivation to enrich the calculus so as to distinguish:

- the rule application failure;
- the empty set of terms.

2.2 A first approach to enriching the ρ-calculus

To distinguish the empty set from rule application failure, we introduce a new symbol $\perp$ (which we assume not to be already in the signature of our calculus) to denote the later. As a consequence, we have to adapt accordingly the definition of calculus operators.

We denote by $\mathcal{S}olution(l \ll^{?}_{\emptyset} t)$ the set of all substitutions obtained when syntactically matching l with t as in [CK01,Cir00]. We consider a meta-application of substitutions denoted by "$\_\langle\!\langle\_\rangle\!\rangle$" and defined by:

$$
\begin{array}{lll}
\mathit{Propagate} & r\langle\!\langle\{\sigma_1, \ldots, \sigma_n\}\rangle\!\rangle \rightsquigarrow \{\sigma_1 r, \ldots, \sigma_n r\} & \\
 & \qquad\qquad\qquad\qquad\quad \mathsf{if}\ n > 0 & \\
\mathit{PropagateEmpty} & r\langle\!\langle\emptyset\rangle\!\rangle \rightsquigarrow \{\perp\} &
\end{array}
$$

Since we have modified the definition of substitution application, the definition of the $Fire$ rule does not need to be changed. So, we obtain the rule defined in Figure 1.

$$\mathit{Fire} \quad [l \to r](t) \rightsquigarrow r\langle\!\langle \mathcal{S}olution(l \ll^{?}_{\emptyset} t)\rangle\!\rangle$$

Fig. 1. $Fire$ rule in the extended ρ-calculus

With respect to the definition of the ρ-calculus we need now to define how the new calculus works regarding the new symbol $\perp$. Therefore, to add the rules of the third part of Figure 2 for describing the propagation of $\perp$ (at this stage of the construction, we do not yet have a strict propagation).

We must emphasize that the side condition $n > 0$ of the rule $FlatBot$ in the third part of Figure 2 is essential so as to avoid $\{\perp\} \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \emptyset$, which would suppress

the wanted distinction! In our calculus, $\{\bot\}$ is a normal form term. Indeed, the semantic of $\{\bot\}$ is the failure term, so we want it to be a result term.

We have added new rules to the ρ-calculus, so let us see what consequences this addition has. To define the `first`, something like the following must be expressed: if $res = \bot$ then evaluate c, which can be expressed at this step of the construction by: $[\bot \to c](res)$, and if $res = \bot$, the successful matching $[\bot \to c](res)$ can be rewritten in $\{c\}$. It seems necessary to have a new symbol function (that is not already in the signature of our calculus) to do a more precise matching, to distinguish the propagation of the failure from its catching.

Example 1. Using the `first` operator (which is discussed in Section 4.1), we can express the evaluation scheme:

if M is evaluated to the failure term
then evaluate the term $P_{failure}$
else evaluate the term P_{normal}.

thanks for example to the term $\big[\texttt{first}\big(\ \bot \to P_{failure}, x \to P_{normal}\big)\big](M)$. But, this has a serious drawback: If M leads to $\{\bot\}$ then, using the rule for the $\bot$ propagation (see the third part of Figure 2) the following reduction may happen: $\big[\texttt{first}\big(\ \bot \to P_{failure}, x \to P_{normal}\big)\big](M) \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \big[\texttt{first}\big(\ \bot \to P_{failure}, x \to P_{normal}\big)\big(\ \bot\ \big) \mapsto\!\!\!\!\rightarrow_{BotOpR} \{\bot\}$.

One of the important properties of the ρ-calculus is its strictness, but this property can not be preserved in our system if we want to allow applications like $[\bot \to c]\ (\bot) \mapsto\!\!\!\!\rightarrow_{Fire} \{c\}$. So, we should improve the behavior of the proposed calculus. Moreover, it must be emphasized that we do not have the strict propagation of the failure with terms having the empty set as a subterm. For instance, $f(\emptyset, \bot) \mapsto\!\!\!\!\rightarrow_{\rho} \emptyset$ is something we would like to rule out in order to have the strict propagation of $\bot$ and not to modify the confluence property since we also have $f(\emptyset, \bot) \mapsto\!\!\!\!\rightarrow_{\rho} \{\bot\}$. To sum up, we must modify our evaluation rules in order to:

- have strictness;
- obtain a confluent calculus under a reasonable evaluation strategy;
- allow failure catching, and as a consequence define the `first`.

2.3 An adapted version of the calculus: the ρ_{ε}-calculus

About the meaning of the empty set As we have seen, in the ρ-calculus an empty set is the representation of both an empty set of terms or of the rule application failure. What we propose now is to give a single meaning to the empty set: to represent a term in itself. Consequently, the application of a term to the empty set should lead to failure (*i.e.* $[v](\emptyset) \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \{\bot\}$) and the application of the empty set to a term should lead too to failure (*i.e.* $[\emptyset](v) \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho} \{\bot\}$). On the other hand, provided all the t_i are in normal form, a term like $f(t_1, \ldots, \emptyset, \ldots, t_n)$ will be considered as a normal form and not rewritable to $\{\bot\}$. This is in particular useful to keep to the empty set its first class status. Moreover, we would like a

ρ-term of the form $u \to \emptyset$ to be in normal form since it allows us to express a void function.

Up to the last rule (Exn), these design choices lead to the rewriting calculus evaluation rules tuning described in Figure 2. Let us now explain the design choices leading to the last rule.

An extension of the calculus to deal with exceptions In particular motivated by our goal to define the `first` operator, we need to express failure catching. Moreover, if we want to control the strictness of failure propagation, we need to encapsulate the failure symbol by a new operator to allow for something like:

$$[operateur(\bot) \to u](operateur(\bot)) \mapsto\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow_\rho u$$

This new operator is denoted exn and its evaluation rule is defined by:

$$Exn \quad exn(t) \rightsquigarrow \{t\} \quad \text{if } \{t\}\downarrow \neq \{\bot\} \text{ and } \mathcal{FV}ar(t\downarrow) = \emptyset$$

The conditions under which this evaluation rule is applicable are similar to the one we have already seen for the evaluation rule $First'$ (Section 2.1): on one hand, we can only decide if a term is a failure when it is in normal form, on the other hand if free variables are still present, they could be instantiated by the evaluation of the context, therefore creating potentially new redexes.

Remark 1. If the evaluation of a term M leads to $\{\bot\}$, then $exn(M) \overset{*}{\mapsto\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow}_\rho exn(\{\bot\})$. It is then natural to allow for the reduction $exn(\{\bot\}) \overset{*}{\mapsto\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow}_\rho \{exn(\bot)\}$ since otherwise we prevent from an evaluation switched by a possible failure in the evaluation of a term (see Example 2 in which the $OpSet$ rule is essential). So, as it can be shown in Figure 2, in the ρ_ε-calculus, we extend the $OpSet$ rule for all ε-functional symbols (*i.e.* for all $h \in \mathcal{F}_\varepsilon \triangleq \mathcal{F} \cup \{exn, \bot\}$, where $\mathcal{F}$ is the signature).

Remark 2. We could have replaced the Exn rule by the Exn' rule:

$$Exn' \quad exn(t) \rightsquigarrow \{t\downarrow\} \quad \text{if } \{t\}\downarrow \neq \{\bot\} \text{ and } \mathcal{FV}ar(t\downarrow) = \emptyset$$

But doing so will keep us away from a small step semantics, since the intrinsic operations at the object level will in this case be executed in a meta-level and the operator at the object level will not contain anymore all the evaluation informations.

Definition of the ρ_ε-calculus The previous design decision lead us to define the ρ_ε-terms and the ε-first-order terms in the following way.

$$\rho_\varepsilon\textbf{-term} \qquad t ::= x \mid f(t, \ldots, t) \mid t \to t \mid [t](t) \mid \{t, \ldots, t\} \mid \bot \mid exn(t)$$

where $x \in \mathcal{X}$ and $f \in \mathcal{F}$.

In the ρ-calculus, first-order terms only are allowed in the left hand side of an abstraction. In spite of this restriction, we obtain a quite expressive calculus which confluence is obtained under a large class of evaluation strategies [CK01,Cir00]. Introducing $\bot$ and exn, we would like to define an extension of the ρ-calculus that keeps these properties. Therefore we extend the terms which we will abstract on (*e.g.* rewrite rule left hand sides) to be as follow:

ε**-first-order term** $\quad t ::= x \mid f(t,\ldots,t) \mid exn(\bot)$

Definition 1. *For $\mathcal{F}$ a set of function symbols and $\mathcal{X}$ a denumerable set of variables, we define the ρ_ε-calculus by:*

- *the subset $\rho_\varepsilon(\mathcal{F},\mathcal{X})$ of ρ_ε-terms which subterms of the form $l \to r$ are such that l is a ε-first-order term;*
- *the higher-order application of substitutions to terms;*
- *the empty theory with the classical syntactic matching algorithm;*
- *the evaluation rules described in Figure 2;*
- *an evaluation strategy $\mathcal{S}$ that fixes the way to apply the evaluation rules.*

Fact 21 *The set $\rho_\varepsilon(\mathcal{F},\mathcal{X})$ is stable by $\mathcal{EVAL}_\varepsilon$.*

Proof. This is simple consequence of the set $\rho_\varepsilon(\mathcal{F},\mathcal{X})$ preservation by the substitution mechanism. This explains in particular why $\mathcal{EVAL}_\varepsilon$ do not have the $SwitchL$ rule of the ρ-calculus .

Example 2. If we want to express the evaluation scheme presented in Example 1, thanks to the exn operator the problem exposed in that example is solved. In fact, in the ρ_ε-term $[\mathtt{first}(exn(\bot) \to P_{failure}, x \to P_{normal})](exn(M))$ the propagation of failure is stopped thanks to the exn operator.

1. if $M \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} \{\bot\}$, we have the following reduction:
$$\begin{array}{ll} & \big[\mathtt{first}\big(exn(\bot) \to P_{failure}, x \to P_{normal}\big)\big]\big(exn(M)\big) \\ \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} & \big[\mathtt{first}\big(exn(\bot) \to P_{failure}, x \to P_{normal}\big)\big]\big(exn(\{\bot\})\big) \\ \mapsto\!\!\!\!\rightarrow_{OpSet} & \big[\mathtt{first}\big(exn(\bot) \to P_{failure}, x \to P_{normal}\big)\big]\big(\{exn(\bot)\}\big) \\ \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} & \{P_{failure}\} \end{array}$$
So, in this case, the initial term leads to $P_{failure}$, which is the wanted behavior.
2. if $M \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} \{M' \downarrow\}$ where $\{M' \downarrow\} \neq \{\bot\}$, we obtain:
$$\begin{array}{ll} & \big[\mathtt{first}\big(exn(\bot) \to P_{failure}, x \to P_{normal}\big)\big]\big(exn(M)\big) \\ \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} & \big[\mathtt{first}\big(exn(\bot) \to P_{failure}, x \to P_{normal}\big)\big]\big(exn(M')\big) \\ \overset{*}{\mapsto\!\!\!\!\rightarrow}_{\rho_\varepsilon} & \{P_{normal}\} \end{array}$$
In this case also, we have the wanted result: P_{normal}.

After introducing this new version of the rewriting calculus that makes a full distinction between rule application failure and the empty set of result, we are going in the next sections to make precise its expressiveness, particularly the behavior of the $exn(\bot)$ pattern, and to make precise in which condition we get a confluent calculus.

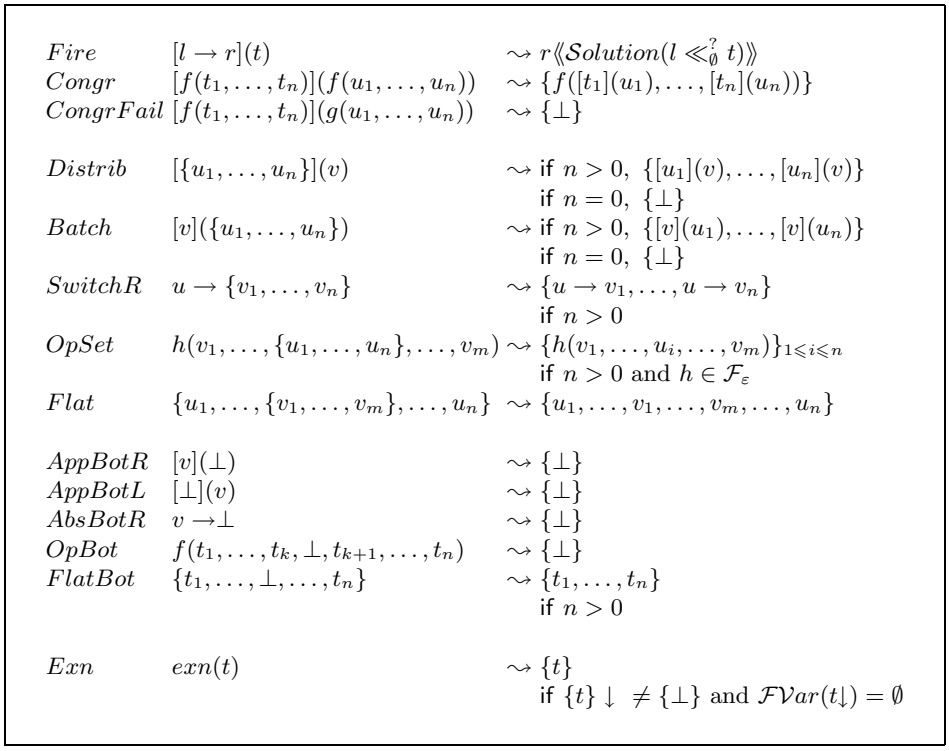


Fig. 2. $\mathcal{EVAL}_\varepsilon$, the evaluation rules of the ρ_ε-calculus

3 On the confluence of the ρ_ε-calculus. The $\rho_{\varepsilon\mathsf{V}}$-calculus

3.1 The non confluence of the ρ_ε-calculus

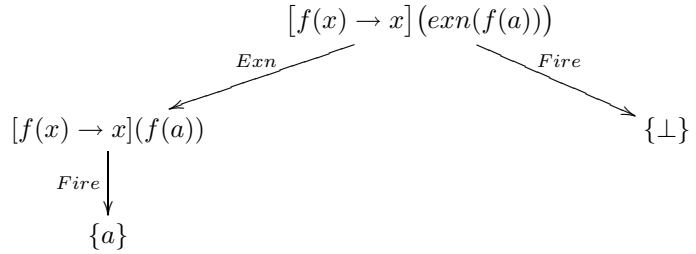
In [Cir00,CK01], it is shown that the ρ-calculus is confluent under a large class of evaluation strategies. The same thing holds here. At first, it may seems easy to generalize but in fact there are two problems that need a particular treatments:

- the Exn rule;
- the multiple ways to obtain $\{\bot\}$.

At the end of this section, we are going to make more precise the $exn(\bot)$ role. Let us begin to show typical examples of confluence failure. Non-confluence is inherited from the confluence failure of the ρ-calculus (described in [Cir00,CK01]) and from specific critical overlaps due to its enrichment and on which we are focusing now.

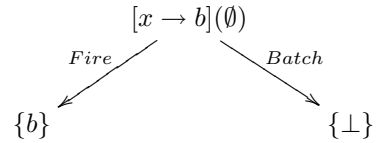
First, we have confluence problems if l and t are matched in order to apply the $Fire$ rule and if t is not Exn-reduced enough (*i.e.* the Exn rule can be applied).

Example 3 (Not evaluated enough terms). The next situation shows that to recover confluence, we must apply the $Fire$ rule to $[l \to r](t)$ only if t is Exn-reduced enough (and indeed we want that t is Exn-reduced not in all positions of t but in positions that corresponds to a ε-functional position in l, *i.e.* in positions that corresponds to a symbol function in $\mathcal{F}_\varepsilon$).



In the ρ-calculus, knowing if a term can be reduced to failure (represented in the ρ-calculus by $\emptyset$) is quite easy since we have only to guarantee that no failure is possible and that $\emptyset$ is not a subterm of t. But in the ρ_ε-calculus, knowing if a term can be reduced to $\{\bot\}$ seems not to be so easy since for example such a term must not contain subterms like $[u](v)$ where u or v can be reduced to $\emptyset$ (see rule $Distrib, Batch$). In fact, if this condition is not verified, we have problems of confluence like the critical overlap of Example 4.

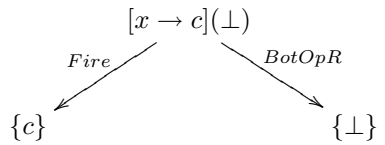
Example 4 (Operand equals to $\emptyset$).



In this example, the problem is indeed more complicated to solve than it seems since we need a test to insure that a term can not be reduced to $\emptyset$, for more details see [Fau01].

A recurrent problem when we construct a calculus is to have a strict calculus (here it means that either the $\bot$ propagation is stopped by exn or the $\bot$ propagation is strict). In the ρ_ε-calculus, if we do not specify any strategy, we do not have a strict propagation of $\bot$ as shown in the following example.

Example 5 (Non-strict propagation of failure).



So, in addition to the difficulty to obtain, thanks to a suitable strategy, a confluent calculus, we are looking for a strategy which allows also for a strict calculus.

The $exn(\bot)$ role The $exn(\bot)$ pattern allows to catch via matching the rule application failure like in $\big[exn(\bot) \to c\big]\big(exn([f(x,y) \to x](g(a,b)))\big)$. Of course, nothing in the ρ_ε-calculus prevents from instantiating a variable by the $exn(\bot)$ pattern. This is something quite natural since it allows us to propagate failure suspicion or to ignore a raised exception like in $[x \to c](exn(M))$.

3.2 How to obtain a confluent calculus? The $\rho_{\varepsilon \mathrm{v}}$-calculus

As seen in the last section, the ρ_ε-calculus is not confluent and not strict. To obtain a confluent calculus, the evaluation mechanism described in $\mathcal{EVAL}_\varepsilon$ should be tamed by a suitable strategy. In addition to the difficulty introduced by the catch-ability, we are looking for a strategy which allows not only for a confluent but also for a strict calculus. This could be described in two ways, either by an independently described strategy or, and this the choice made in [Fau01], by conditions directly added to the $Fire$ and Exn rules. Here, we do not describe these conditional rules and we refer to [Fau01] for the detailed approach. Actually, instead of giving technical conditions to apply the $Fire$ and Exn rules, we are going to define a call by value strategy and we are going to show that the ρ_ε-calculus is confluent when evaluated using a call by value strategy, a calculus named $\rho_{\varepsilon \mathrm{v}}$-calculus.

Intuitively, in the ρ_ε-calculus, a value is a normal form term containing no free variable and no set (except the empty set as the argument of a function):

$$\textbf{value} \qquad v ::= c \;\mid\; f(v,\ldots,v) \;\mid\; f(v,\ldots,\emptyset,\ldots,v) \;\mid\; v \to v \;\mid\; exn(\bot)$$

(where $c \in \mathcal{F}_0$ and $f \in \cup_{n\geq 1}\mathcal{F}_n$)
We can define the $\rho_{\varepsilon \mathrm{v}}$-calculus either by using the classical syntactic algorithm matching or by adding a new rule to it. Actually, although it is not necessary for the confluence, it will be judicious that during the matching no term can be instanced by $\emptyset$. So, as it can be shown in Figure 3, we add the $EmptyInstanciation$ rule so as to avoid this instantiation.

$Decomposition$	$(f(t_1,\ldots,t_n) \ll^?_\emptyset f(t'_1,\ldots,t'_n)) \wedge P$	$\mapsto\!\!\!\!\rightarrow \bigwedge_{i=1\ldots n} t_i \ll^?_\emptyset t'_i \wedge P$
$SymbolClash$	$(f(t_1,\ldots,t_n) \ll^?_\emptyset g(t'_1,\ldots,t'_m)) \wedge P$	$\mapsto\!\!\!\!\rightarrow \mathbf{F}$ if $f \neq g$
$MergingClash$	$(x \ll^?_\emptyset t) \wedge (x \ll^?_\emptyset t') \wedge P$	$\mapsto\!\!\!\!\rightarrow \mathbf{F}$ if $t \neq t'$
$EmptyInstanciation$	$(x \ll^?_\emptyset \{\}) \wedge P$	$\mapsto\!\!\!\!\rightarrow \mathbf{F}$
$SymbolVariableClash$	$(f(t_1,\ldots,t_n) \ll^?_\emptyset x) \wedge P$	$\mapsto\!\!\!\!\rightarrow \mathbf{F}$ if $x \in \mathcal{X}$

Fig. 3. The call by value syntactic matching

In the $\rho_{\varepsilon \mathrm{v}}$-calculus, we are going to apply the $Fire$ and Exn rules, only to values. Assume that l, r, t are ρ_ε-terms and v are values, we define $Fire_v, Exn_v$ by:

$$\begin{array}{ll} Fire_v\ [l \to r](v) & \rightsquigarrow r\langle\!\langle \mathcal{S}olution(l \ll^?_\emptyset v)\rangle\!\rangle \\ Exn_v\ exn(v) & \rightsquigarrow v \end{array}$$

So, now we can define more precisely the $\rho_{\varepsilon \mathrm{v}}$-calculus:

Definition 2. *The $\rho_{\varepsilon \mathrm{v}}$-calculus is defined as the ρ_ε-calculus except that we replace the $Fire$ and Exn rules by the $Fire_v$ and Exn_v rules and that $\mathcal{S}olution(l \ll^?_\emptyset v)$ is computed by the matching algorithm of Figure 3.*

In [Fau01], it is shown that this calculus is confluent and strict (since $\perp$ is not a value):

Theorem 1. *The $\rho_{\varepsilon \mathrm{v}}$-calculus is a confluent and strict calculus.*

This result is fundamental in the following since we are going to see that it is thanks to the confluence that we can express the `first` in the $\rho_{\varepsilon \mathrm{v}}$-calculus. Thus, in the following section, we show that the $\rho_{\varepsilon \mathrm{v}}$-calculus allows us to express the `first` operator and consequently that we obtain a confluent calculus in which the `first` can be expressed.

4 Expressiveness of the $\rho_{\varepsilon \mathrm{v}}$-calculus

4.1 The `first` operator in the $\rho_{\varepsilon \mathrm{v}}$-calculus

Originally, the ρ_ε-calculus was introduced to obtain a calculus in which the `first` can be simply expressed. All over the construction of the ρ_ε-calculus, we aim at eliminating all arguments that help us to think that the `first` can not be expressed in it. We are going to show that we can indeed find a shallow encoding of it in the $\rho_{\varepsilon \mathrm{v}}$-calculus. So, we define a term to express it, afterwards we will prove that the given definition is valid and finally we give some examples.

As we have just seen, the `first` is an operator whose role is to select between its arguments the first one that, applied to a given ρ-term, does not evaluate to $\{\perp\}$. This is something quite difficult to express in the ρ-calculus. So, to solve this problem, we propose to represent the term $[\texttt{first}(t_1, \ldots, t_n)](r)$ by a set of n terms, noted $\{u_1, \ldots, u_n\}$. Each time, no more than one (say u_l) of these terms can not be reduced to $\{\perp\}$. So, after reductions, thanks to the $FlatBot$ rule, the initial set of n terms is reduced to a singleton set consisting of the normal form of u_l, which is going to represent the result of $[\texttt{first}(t_1, \ldots, t_n)](r)$. To express the `first` in the ρ_ε-calculus, we are going to use its two main trumps: the matching and the failure catching. Each u_i is expressed in accordance with the $i-1$ first terms: if all u_j $(j < i)$ are reduced to $\{\perp\}$, then u_i leads to $[t_i](r)$ else u_i leads to $\{\perp\}$ thanks to a rule application failure. So, we define each u_i by

$$
\begin{array}{l}
u_1 = [t_1](x) \\
u_2 = \big[exn(\bot) \to [t_2](x)\big]\big(exn(u_1)\big) \\
u_3 = \big[exn(\bot) \to [exn(\bot) \to [t_3](x)](exn(u_2))\big]\big(exn(u_1)\big) \\
u_4 = \big[exn(\bot) \to [exn(\bot) \to [exn(\bot) \to [t_4](x)](exn(u_3))](exn(u_2))\big]\big(exn(u_1)\big) \\
\vdots \\
u_n = [exn(\bot) \to [exn(\bot) \to [\ldots \to [t_n](x)](exn(u_{n-1}))\ldots](exn(u_2))](exn(u_1))
\end{array}
$$

To get the right definition, one should replace in the expression of u_i, all u_j ($j < i$) by their own definition but, such extended formulae would be rather tedious to read. We define $MyFirst$ by:

$$MyFirst(t_1, \ldots, t_n) \triangleq x \to \{u_1, \ldots, u_n\}$$

By $\texttt{first}(t_1, \ldots, t_n)$, we denote precisely the `first` operator as defined in Section 2.1 by the two rules $First', First''$ and we denote by $MyFirst(t_1, \ldots, t_n)$ the operator purely defined in the ρ_ε-calculus just above.

For each u_i, a matching is done to know if u_j (for all u_j, $j < i$) can be reduced to the failure term or not. It must be noticed that this definition is not valid in a non-confluent calculus. In fact, if $MyFirst$ is used in this kind of calculus, it may happen that there exists i_0 such that u_{i_0} can be evaluated to a term $v_{i_0} \neq \{\bot\}$ thanks to a first reduction and to $\{\bot\}$ thanks to a second one. In this case, we can have for example a first reduction used to evaluate u_{i_0} that leads to $\{\bot\}$, but in one of the u_k ($k > i_0$) we can have a different reduction of the same term u_{i_0} that leads to $v_{i_0} \neq \{\bot\}$ and so u_k is reduced as if u_{i_0} is not evaluated to $\{\bot\}$, whereas, here, it is. So, we obtain a set of terms which do not necessary have the property that no more than one of these can not be reduced to $\{\bot\}$. Thus, in this section, we are going to work in a confluent calculus: the $\rho_{\varepsilon \mathrm{v}}$-calculus (although we can work in every confluent ρ_ε-calculus). Let us show the validity of the definition of $MyFirst$.

Lemma 1. *In the $\rho_{\varepsilon \mathrm{v}}$-calculus,*

- *if for all i, $[t_i](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{\bot\}$, then $[MyFirst(t_1, \ldots, t_n)](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{\bot\}$ and $[\texttt{first}(t_1, \ldots, t_n)](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{\bot\}$.*
- *if it exists l such that for all $i \leq l-1$ $[t_i](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{\bot\}$ and $[t_l](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{v_l\} \downarrow \neq \{\bot\}$ where $\mathcal{FV}ar(v_l) = \emptyset$ then $[MyFirst(t_1, \ldots, t_n)](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{v_l\} \downarrow$ and $[\texttt{first}(t_1, \ldots, t_n)](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{v_l\} \downarrow$.*

Proof. See [Fau01].

If it exists l such that for all $i \leq l-1$ $[t_i](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{\bot\}$ and $[t_l](r) \overset{*}{\twoheadrightarrow}_{\rho_{\varepsilon v}} \{v_l\}\downarrow \neq \{\bot\}$ with $\mathcal{FV}ar(v_l) \neq \emptyset$, since a term with no free variable is not a value, we can not "really evaluate" $[MyFirst(t_1, \ldots, t_n)](r)$ and so, the evaluation is suspended (as it is for the `first`). So, *w.r.t.* terms with free variables, this result is also coherent. To illustrate this, we give the following example:

Example 6.
$$\begin{array}{ll} & x \to [MyFirst\big(y \to [x](y), y \to c\big)](b) \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & x \to \{[x](b), [exn(\bot) \to [y \to c](b)](exn\big([x](b)\big))\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{x \to [x](b), x \to [exn(\bot) \to c](exn\big([x](b)\big))\} \end{array}$$

In the $\rho_{\varepsilon \mathrm{v}}$-calculus this result is in normal form because we can not continue the evaluation since the variable x needs to be instantiated. Let us note that it is not a drawback since we have the same behavior if we consider the `first` defined by the rules $First'$ and $First''$ given in Section 2.1.

Thanks to the two above lemmas, we get our main expressiveness result:

Theorem 2. *The `first` is simply expressible in the $\rho_{\varepsilon \mathrm{v}}$-calculus.*

Now, we will denote by $\texttt{first}(t_1, \ldots, t_n)$ the term $MyFirst(t_1, \ldots, t_n)$. The above theorem is significant since this question about the expressiveness of the `first` in the ρ-calculus has arisen since the introduction of the ρ-calculus. Moreover, we construct not only a calculus in which the `first` is expressible, but we allow a mechanism of exceptions as it can be shown in all examples of Section 4.2. Moreover, it is important to mention that this result (a confluent calculus in which the `first` can be expressed) is proved here for the first time since the confluence of the ρ^{1st}-calculus (the ρ-calculus doped with the `first`) is still an open question.

In the next example, we provide an example of two reductions of the same term illustrating the confluence of the proposed calculus.

Example 7. A first reduction:

$$\begin{array}{ll} & \big[x \to [\texttt{first}\big(y \to [y](x),\ y \to c\big)](b)\big]\big(a\big) \\ \triangleq & \big[x \to \big\{[y \to [y](x)](b), \\ & \qquad [exn(\bot) \to [y \to c](b)](exn([y \to [y](x)](b)))\big\}\big]\big(a\big) \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \big[x \to \{[b](x),\ [exn(\bot) \to [y \to c](b)](exn([b](x)))\}\big]\big(a\big) \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{\big[b\big]\big(a\big)\ ,\ [exn(\bot) \to [y \to c](b)](exn([b](a)))\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{\{\bot\},\ [y \to c](b)\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{c\} \end{array}$$

An other possible reduction could be:

$$\begin{array}{ll} & \big[x \to [\texttt{first}(y \to [y](x),\ y \to c)](b)\big]\big(a\big) \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \big[\texttt{first}(y \to [y](a),\ y \to c)\big]\big(b\big) \\ \triangleq & \{[y \to [y](a)](b),\ [exn(\bot) \to [y \to c](b)](exn([y \to [y](a)](b)))\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{\bot,\ \{c\}\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\longmapsto\!\!\!\!\!\rightarrow}} & \{c\} \end{array}$$

In Section 2.3, we give a term that allows us to encode an evaluation scheme switched by the result of the evaluation of one term (failure or not). To express this term, we use the `first` operator. As we have just seen, this operator is expressible in the $\rho_{\varepsilon \mathrm{v}}$-calculus. So, let us show in this example how it does really work:

Example 8.

$$\begin{array}{ll} & \big[\texttt{first}(exn(\bot) \to P_{failure},\ x \to P_{normal})\big]\big(exn(M)\big) \\ \triangleq & \big\{\big[exn(\bot) \to P_{failure}\big]\big(exn(M)\big)\ , \\ & \quad \big[exn(\bot) \to [x \to P_{normal}](exn(M))\big]\big([exn(\bot) \to P_{failure}](exn(M))\big)\ \big\} \end{array}$$

The evaluation strategy (call by value) forces to begin by the evaluation of M. We are going to distinguish two cases:

1. if $M \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} \{\bot\}$, we have the following reduction:

$$\begin{array}{ll} & \big\{\big[exn(\bot) \to P_{failure}\big]\big(exn(M)\big)\ , \\ & \quad \big[exn(\bot) \to [x \to P_{normal}](exn(M))\big]\big([exn(\bot) \to P_{failure}](exn(M))\big)\ \big\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} & \big\{\{P_{failure}\},\ \{\bot\}\big\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} & \{P_{failure}\} \end{array}$$

   So, in this case, the initial term leads to $P_{failure}$, which is the wanted result.
2. if $M \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} \{M' \downarrow\}$ where $\{M' \downarrow\} \neq \{\bot\}$, we obtain:

$$\begin{array}{ll} & \big\{\big[exn(\bot) \to P_{failure}\big]\big(exn(M)\big)\ , \\ & \quad \big[exn(\bot) \to [x \to P_{normal}](exn(M))\big]\big([exn(\bot) \to P_{failure}](exn(M))\big)\ \big\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} & \big\{\{\bot\},\ [x \to P_{normal}](exn(M))\big\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} & \big\{\{[x \to P_{normal}](M')\}\big\} \\ \overset{*}{\underset{\rho_{\varepsilon v}}{\twoheadrightarrow}} & \{P_{normal}\} \end{array}$$

   In this last case also, we have the wanted result: P_{normal}.

These examples enlighten the expressiveness of the $\rho_{\varepsilon \mathrm{v}}$-calculus. In the following section, we are going to see that we can for example:

- express a failure catching as precise as desired;
- express normalization strategy;
- fully allow for an exception mechanism.

4.2 Examples

In the Example 2, we have seen that the $\rho_{\varepsilon \mathrm{v}}$-calculus allows to switch the evaluation mechanism according to a possible failure in the evaluation of a term. In the following example we are going to see that we can switch the evaluation according to a failure catching as precise as desired.

Example 9. Let us consider the evaluation scheme:
if the two arguments of f are evaluated to the failure
then evaluate $P_{failure1\&2}$
else if the first argument of f is evaluated to the failure
then evaluate $P_{failure1}$
else if the second argument of f is evaluated to the failure
then evaluate $P_{failure2}$

else evaluate P_{normal}

It can be expressed in the $\rho_{\varepsilon \mathrm{v}}$-calculus by the following term (one should replace `first` by its given definition):

$$\Big[\ \texttt{first}\ \Big(\begin{array}{ll} f\big(exn(\bot), exn(\bot)\big) & \to P_{failure1\&2}, \\ f\big(exn(\bot), x\big) & \to P_{failure1}, \\ f\big(x, exn(\bot)\big) & \to P_{failure2}, \\ f\big(x, y\big) & \to P_{normal} \end{array}\ \Big)\ \Big] \quad \Big(f\big(exn(M), exn(N)\big)\Big)$$

So, we can switch the evaluation scheme according to the possible evaluation failure of two arguments of a function. It is clear that a similar term can be constructed for every function.

The next example shows how to express in the $\rho_{\varepsilon \mathrm{v}}$-calculus the exception mechanism of ML [Mil78,WL96], when no named exception is considered.

Example 10. The following ML program:

```
try
 x/y;
 P1
with Div_By_Zero -> P2
```

is expressed in the $\rho_{\varepsilon \mathrm{v}}$-calculus, as:

$$\big[\texttt{first}\big(exn(\bot) \to P2, x \to P1\big)\big]\big(exn(x \not/_{\rho} y)\big)\big)$$

where $\not/_{\rho}$ is an encoding of x/y in the rewriting calculus. Similarly, since we can ignore a raised exception (see Section 3.1), we can express the ML program

```
try
 P1
with _ -> P2
```

by the ρ-term $[x \to P2](exn(P1))$.

Example 11. One of the `first`'s strong interest is to allow a full and explicit encoding of normalization strategy. For instance, an innermost normalization operator can be build, thanks to the `first` [Cir00,CK01]. This can be expressed, as follows:

– A term traversal operator for all operators f_i in the signature:

$$\begin{array}{rl} \Phi(r) \triangleq first(& f_1(r, id, \ldots, id), \ldots, f_1(id, \ldots, id, r), \\ & \ldots, \\ & f_m(r, id, \ldots, id), \ldots, f_m(id, \ldots, id, r)) \end{array}$$

Using Φ we get for example:
$[\Phi(a \to c)](f(a, b)) \twoheadrightarrow_{\rho_\varepsilon} \{f(c, b)\}$

$[\Phi(a \to b)](c) \twoheadrightarrow_{\rho_\varepsilon} \{\bot\}$

– The *BottomUp* operator

$$Once_{bu}(r) \triangleq [\Theta](H_{bu}(r)) \; with \; H_{bu}(r) \triangleq f \to (x \to [first(\Phi(f), r)](x))$$

where Θ is Turing's fixed-point combinator expressed in the ρ-calculus:

$$\Theta = [A](A) \; with \; A = x \to (y \to [y]([[x](x)](y)))$$

For instance we have: $[Once_{bu}(a \to b)](f(a, g(a)) \twoheadrightarrow_{\rho_\varepsilon} \{f(b, g(a))\})$

– A repetition operator

$$repeat * (r) \triangleq [\Theta](J(r)) \; with \; J(r) \triangleq f \to (x \to [first(r; f, \; id)](x))$$

This can be used like in $[repeat * (\{a \to b, b \to c\})](a) \twoheadrightarrow_{\rho_\varepsilon} \{c\}$.

– Finally we get the inner-most normalization operator:

$$im(r) \triangleq repeat * (Once_{bu}(r))$$

It allows us to reduce $f(a, g(a))$ to its inner-most normal form according to the rewriting system $\mathcal{R} = \{a \to b, f(x, g(x)) \to x\}$. For instance, $[im(\mathcal{R})](f(a, g(a))) \twoheadrightarrow_{\rho_{\varepsilon v}} \{b\}$. Notice that nowhere we need to assume confluence of $\mathcal{R}$: considering the rewrite system $\mathcal{R}' = \{a \to b, a \to c, f(x, x) \to x\}$, we have $[im(\mathcal{R}')](f(a, a)) \twoheadrightarrow_{\rho_{\varepsilon v}} \{b, f(c, b), f(b, c), c\}$.

Expressing normalization strategy becomes explicit and reasonably easy since in ρ-calculus, terms rules, rule applications and therefore strategies are considered at the object level.

So, it is quite natural to ask if we can express how applying these rules. Since the `first` is expressible in the $\rho_{\varepsilon \mathrm{v}}$-calculus, we can express all above operators and we obtain as a corollary:

Corollary 1. *The operators im (and om) describing the innermost (resp. outermost) normalization can be expressed in the $\rho_{\varepsilon \mathrm{v}}$-calculus. Moreover, given a rewriting theory and two first order ground terms (in the sense of rewriting) such that t is normalized to $t{\downarrow}$ w.r.t. the set of rewrite rules $\mathcal{R}$, the term $[im(\mathcal{R})](t)$ is $\rho_{\varepsilon v}$-reduced to a set containing the term $t{\downarrow}$.*

Notice that in this situation, a rewrite system is innermost confluent if and only if $[im(\mathcal{R})](t)$ is a singleton. A term is in inner-most normal form if this singleton is $\{\bot\}$.

Notice also that, following [CK01], normalized rewriting [DO90] using a conditional rewrite rule of the form $l \to r$ if c can be simply expressed as the ρ-term $l \to [\mathrm{True} \to r]([im(\mathcal{R})](c))$ and therefore that conditional rewriting is simply expressible in the $\rho_{\varepsilon \mathrm{v}}$-calculus.

5 Conclusion and further work

By the introduction of an exception mechanism in the rewriting calculus, we have solved the simple expression problem of the `first` operator in a confluent calculus. Consequently, this solves the open problem of the doped calculus confluence.

Motivated by this expressiveness question, we have indeed elaborated a powerful variation of the rewriting calculus which is useful first for theorem proving when providing general proof search strategies in a semantically well founded language and second as a useful programming paradigm for rule based languages. It has in particular the main advantage to bring the exception paradigm uniformly at the level of rewriting.

This could be extended in particular by allowing for named exceptions. Another challenging question is now to express, may be in the $\rho_{\varepsilon\mathrm{v}}$-calculus, strategy operators like the "don't care" one used for example in the ELAN language.

Acknowledgement. We thank Daniel Hirschkoff for his strong interest and support in this work, Horatiu Cirstea and Luigi Liquori for many stimulating discussions on the rewriting calculus and its applications and for their constructive remarks on the ideas developped here.

References

[BGG+92] R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. V. Tassel. Experience with embedding hardware description languages in HOL. In V. Stavridou, T. F. Melham, and R. T. Boute, editors, *Proceedings of the IFIP TC10/WG 10.2 International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience*, volume A-10 of *IFIP Transactions*, pages 129–156, Nijmegen, The Netherlands, June 1992. North-Holland/Elsevier.

[BKK+98] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume15.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science. Report LORIA 98-R-316.

[Cir00] H. Cirstea. *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, France, October 2000.

[CK01] H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[CKL01a] H. Cirstea, C. Kirchner, and L. Liquori. The Matching Power. In V. van Oostrom, editor, *Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, May 2001. Springer-Verlag.

[CKL01b] H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. In F. Honsell, editor, *Foundations of Software Science and Computation Structures*, volume 2030 of *Lecture Notes in Computer Science*, pages 166–180, Genova, Italy, April 2001.

[DK00] H. Dubois and H. Kirchner. Objects, rules and strategies in `ELAN`. In *Proceedings of the second AMAST workshop on Algebraic Methods in Language Processing, Iowa City, Iowa, USA*, May 2000.

[DO90] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[Fau01] G. Faure. Etude des propriétés du calcul de réécriture: du $\rho_{\emptyset}$-calcul au ρ_{ε}-calcul. Rapport, LORIA and ENS-Lyon, September 2001. `http://www.loria.fr/~ckirchne/=rho/rho.html`.

[Mil78] R. Milner. A theory of type polymorphism in programming. *JCSS*, 17:348–375, 1978.

[Ngu01] Q.-H. Nguyen. Certifying term rewriting proof in elan. In M. van den Brand and R. Verma, editors, *Proc. of RULE'01*, volume 59. Elsevier Science Publishers B. V. (North-Holland), September 2001. Available at http://www.elsevier.com/locate/entcs/volume59.html.

[WL96] Weiss and Leroy. *Le langage Caml*. InterEditions, 1996.