Towards a sharing strategy for the graph rewriting calculus

P. Baldan[a] and C. Bertolissi[b] and H. Cirstea[c] and C. Kirchner[d]

[a] *Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy*

[b] *LIF-Université de Provence, Marseille, France*

[c] *LORIA-UHP Nancy 1, France*

[d] *INRIA-LORIA, Nancy, France*

Abstract

The graph rewriting calculus is an extension of the ρ-calculus, handling graph like structures rather than simple terms. The calculus over terms is naturally generalized by using unification constraints in addition to the standard ρ-calculus matching constraints. The transformations are performed by explicit application of rewrite rules as first class entities. The possibility of expressing sharing and cycles allows one to represent and compute over regular infinite entities.
We propose in this paper a reduction strategy for the graph rewriting calculus which aims at maintaining the sharing information as long as possible in the terms. The corresponding reduction relation is shown to be confluent and complete *w.r.t.* the small-step semantics of the graph rewriting calculus.

1 Introduction

Main interest for term rewriting stem from functional and rewrite based languages as well as from theorem proving. In particular, we can describe the behaviour of a functional or rewrite based program by analyzing some properties of the associated term rewriting system. In this framework, terms are often seen as trees but in order to improve the efficiency of the implementation of such languages, it is of fundamental interest to think and implement terms as graphs [BvEG+87]. In this case, the possibility of sharing subterms allows to save space (by using multiple pointers to the same subterm instead of duplicating the subterm) and to save time (a redex appearing in a shared subterm will be reduced at most once and equality tests can be done in constant time when the sharing is maximal).

Graph rewriting is a useful technique for the optimization of functional and declarative languages implementations [PJ87]. Moreover, the possibility to define cycles leads to an increased expressive power that allows one to represent easily regular infinite data structures. Cyclic term graph rewriting has been widely studied, both from an operational [BvEG+87,AK96] and from a categorical/logical point of view [CG99] (see [SPvE93] for a survey on term graph rewriting).

The graph rewriting calculus, or ρ_{g}-calculus, introduced in [Ber05], is a common generalization of the cyclic λ-calculus [AK97] and the ρ-calculus [CK01], providing a framework where pattern matching, graphical structures and higher-order capabilities are primitive. The ρ_{g}-calculus deals with cyclic terms with bound variables and can express vertical sharing as well as horizontal sharing by means of a list of recursion equations. In the ρ_{g}-calculus computations related to the matching are made explicit and performed at the object-level. The calculus, under suitable linearity constraints for patterns, has been shown to be confluent [BBCK07] and expressive enough for simulating cyclic λ-calculus and term-graph rewriting.

In view of a future implementation, we are interested in improving the efficiency of the ρ_{g}-calculus. To this aim we present a reduction strategy aimed at keeping the sharing information as long as possible in ρ_{g}-calculus terms. In the ρ_{g}-calculus the loss of sharing is caused by the application of the substitution rules, which allow to create copies of (sub)terms of a ρ_{g}-calculus term. Indeed, during the computation, some loss of sharing is unavoidable, for example for making a rule application explicit or for solving a matching constraint. However, a strategy which suitably restricts the application of the substitution rules can avoid some useless loss of sharing, leading to more compact normal forms. The strategy should allow to perform essentially the same reductions to normal form as in the unconstrained calculus, in the sense that the normal form of a term with respect to the strategy (when it exists) should be the same as in the original calculus, up to sharing.

Indeed, we will show that, under suitable linearity constraints, the proposed strategy is correct and complete with respect to the reduction relation of the ρ_{g}-calculus. Additionally, we will show that the reduction relation of the ρ_{g}-calculus induced by such strategy is confluent.

The paper is organized as follows. In the first section we review the graph rewriting calculus. Section 3 describes the reduction strategy *SharingStrat* proposed for preserving sharing in ρ_{g}-calculus terms. In Section 4 we show that *SharingStrat* is sound and complete with respect to the small step semantics of the ρ_{g}-calculus. Moreover, along the lines of the proof of confluence for the ρ_{g}-calculus, we show that the ρ_{g}-calculus with *SharingStrat* is confluent. We conclude in Section 5 by presenting some perspectives of future work.

2 The graph rewriting calculus

The syntax of the ρ_{g}-calculus is presented in Fig. 1. As in the plain ρ-calculus, λ-abstraction is generalized by a rule abstraction $P \to G$, where P is in general an arbitrary term. There are two different application operators: the functional application operator, denoted simply by concatenation, and the constraint application operator, denoted by "$\_\ [\_]$". Terms can be grouped together into *structures* built using the operator "$\_ \wr \_$". This operator is useful for representing the (non-deterministic) application of a set of rewrite rules and consequently, the non-deterministic results. For example, the nondeterministic application of one of the rules in $\{a \to b,\ a \to c\}$ to the term a can be written $(a \to b \wr a \to c)\ a$. This term, as it will become clearer after the formal definition of the semantics of the calculus (see Figure 2), reduces to $(a \to b)\ a \wr (a \to c)\ a$ and then to $b \wr c$. Note

Terms			**Constraints**		
$\mathcal{G}, \mathcal{P} ::=$	$\mathcal{X}$	(Variables)	$\mathcal{C} ::=$	ϵ	(Empty constraint)
	$\mid \mathcal{K}$	(Constants)		$\mid \mathcal{X} = \mathcal{G}$	(Recursion equation)
	$\mid \mathcal{P} \twoheadrightarrow \mathcal{G}$	(Abstraction)		$\mid \mathcal{P} \ll \mathcal{G}$	(Match equation)
	$\mid \mathcal{G}\,\mathcal{G}$	(Functional application)		$\mid \mathcal{C}, \mathcal{C}$	(Conjunction)
	$\mid \mathcal{G} \wr \mathcal{G}$	(Structure)			
	$\mid \mathcal{G}\ [\mathcal{C}]$	(Constraint application)			

Fig. 1. Syntax of the ρ_{g}-calculus

that the calculus is untyped, but type systems, in the style of those introduced for the ρ-calculus in [BCKL03,Wac05], would be conceivable.

In the ρ_{g}-calculus constraints are conjunctions (built using the operator "$\_,\_$") of match equations of the form $\mathcal{P} \ll \mathcal{G}$ and recursion equations of the form $\mathcal{X} = \mathcal{G}$. The empty constraint is denoted by ϵ. The operator "$\_,\_$" is supposed to be associative, commutative, with ϵ as neutral element.

We assume that the application operator associates to the left, while the other operators associate to the right. To simplify the syntax, operators have different priorities. Here are the operators ordered from higher to lower priority: "$\_\ \_$", "$\_ \twoheadrightarrow \_$", "$\_ \wr \_$", "$\_\ [\_]$" , "$\_ \ll \_$", "$\_ = \_$" and "$\_,\_$".

The symbols $G, H, P \ldots$ range over the set $\mathcal{G}$ of terms, $x, y, \ldots$ range over the set $\mathcal{X}$ of variables, $a, b, \ldots$ range over a set $\mathcal{K}$ of constants. The symbols $E, F, \ldots$ range over the set $\mathcal{C}$ of constraints.

We call *algebraic* the terms of the form $(((f\ \ G_1)\ \ G_2)\ldots)\ \ G_n$, with $f \in \mathcal{K}$, $G_i \in \mathcal{X} \cup \mathcal{K}$ or G_i algebraic for $i = 1 \ldots n$, and we usually denote them by $f(G_1, G_2, \ldots, G_n)$.

We denote by $\bullet$ (black hole) a constant, already introduced in [AK96] using the equational approach and also in [Cor93] using the categorical approach, to give a name to "undefined" terms that correspond to the expression $x\ [x = x]$ (self-loop). The notation $x =_\circ x$ is an abbreviation for the sequence $x = x_1, \ldots, x_n = x$. We use the symbol $\mathsf{Ctx}[\_]$ for a context with exactly one hole $\_$. We say that a ρ_{g}-term is *acyclic* if it contains no sequence of constraints of the form $\mathsf{Ctx}_0[x_0] \lll \mathsf{Ctx}_1[x_1], \mathsf{Ctx}_2[x_1] \lll \mathsf{Ctx}_3[x_2], \ldots, \mathsf{Ctx}_m[x_n] \lll \mathsf{Ctx}_{m+1}[x_0]$, with $n, m \in \mathbb{N}$ and $\lll \in \{=, \ll\}$. A sequence of this kind is called a *cycle*.

For the purposes of this paper we restrict to left-hand sides of abstractions and match equations that are *acyclic*, *algebraic terms* in *normal form*. The set of all these terms, called *patterns*, is denoted by $\mathcal{P}$. For instance, the ρ_{g}-term $f(y)\ [y = g(y)] \twoheadrightarrow a$ is not allowed since the abstraction has a cyclic left-hand side.

A ρ_{g}-term is called *well-formed* if each variable occurs at most once as left-hand side of a recursion equation. All the ρ_{g}-terms considered in the sequel will be implicitly assumed to be well-formed.

The notions of free and bound variables of ρ_{g}-terms take into account the three binders of the calculus: abstraction, recursion and match. Intuitively, variables on the left hand-side of any of these operators are bound by the operator. As usual, we work modulo *α-conversion*. The set of free variables of a ρ_{g}-term G is denoted by $\mathcal{FV}(G)$. A variable in a term G is called *active*, or in *active position*, if it appears free in the left-hand side of an application occurring in G. Moreover, given a constraint

$\mathcal{C}$ we will refer to the set $\mathcal{DV}(\mathcal{C})$, of variables "defined" in $\mathcal{C}$. This set includes, for any recursion equation $x = G$ in $\mathcal{C}$, the variable x and for any match $P \ll G$ in $\mathcal{C}$, the set of free variables of P. For a formal definition, see [BBCK05].

Finally, in order to ensure the confluence of the calculus, we will assume all patterns to be linear. Roughly, a pattern is called linear if each variable occurs free at most once in the pattern.

Definition 2.1 (Linear ρ_{g}-calculus) *The class of (algebraic)* linear patterns *is defined as follows:*

$$\mathcal{L} \ ::= \ \mathcal{X} \ \mid \ \mathcal{K} \ \mid \ (((\mathcal{K}\,\mathcal{L}_0)\,\mathcal{L}_1)\ldots)\,\mathcal{L}_n \ \mid \ \mathcal{L}_0\ [\mathcal{X}_1 = \mathcal{L}_1, \ldots, \mathcal{X}_n = \mathcal{L}_n]$$

where we assume that $\mathcal{FV}(\mathcal{L}_i) \cap \mathcal{FV}(\mathcal{L}_j) = \emptyset$ for $i \neq j$. A constraint $[L_1 \lll G_1, \ldots, L_n \lll G_n]$, where $\lll \in \{=, \ll\}$, is linear *if all patterns $L_1, \ldots, L_n$ are linear and $\mathcal{FV}(L_i) \cap \mathcal{FV}(L_j) = \emptyset$, $i \neq j$. The* linear ρ_{g}-calculus *is the calculus where all the patterns in the left-hand side of abstractions and all constraints are linear.*

In this paper we will focus on the linear ρ_{g}-calculus, hence the qualification "linear" will be often omitted, and the involved patterns and constraints will be assumed to be linear unless stated otherwise.

We define next an order over variables bound by a match or an equation. This order will be later used in the definition of the substitution rule of the calculus, which will allow one only "upward" substitutions, a constraint which is essential for the confluence of the calculus (see [BBCK07]). We denote by $\leq$ the least pre-order on recursion variables such that $x \geq y$ if $x = \mathsf{Ctx}[y]$ appears in the list of constraints for some context $\mathsf{Ctx}[\_]$. The equivalence induced by the pre-order is denoted $\equiv$ and we say that x and y are cyclically equivalent ($x \equiv y$) if $x \geq y \geq x$ (they lie on a common cycle). We write $x > y$ if $x \geq y$ and $x \not\equiv y$.

Example 2.2 [Some ρ_{g}-terms]

(i) In the rule $(2 * f(x)) \twoheadrightarrow ((y + y)\ [y = f(x)])$ the sharing in the right-hand side avoids the copying of the object instantiating $f(x)$, when the rule is applied to a ρ_{g}-term.

(ii) The ρ_{g}-term $x\ [x = cons(0, x)]$ represents an infinite list of zeros.

(iii) The ρ_{g}-term $f(x, y)\ [x = g(y), y = g(x)]$ is an example of twisted sharing that can be expressed using mutually recursive constraints (to be read as a `letrec` construct). We have that $x \geq y$ and $y \geq x$, hence $x \equiv y$.

The complete set of evaluation rules of the ρ_{g}-calculus is presented in Fig. 2. As in the plain ρ-calculus, in the ρ_{g}-calculus the application of a rewrite rule to a term is represented as the application of an abstraction. A redex can be activated using the ρ rule in the Basic rules, which creates the corresponding matching constraint. The computation of the substitution which solves the matching is then performed explicitly by the Matching rules and, if the computation is successful, the result is a recursion equation added to the list of constraints of the term. This means that the substitution is not applied immediately to the term but it is kept in the environment for a delayed application or for deletion if useless, as expressed by the Graph rules.

More precisely, the first two rules ρ and δ come from the ρ-calculus. For each of

BASIC RULES:

$$\begin{array}{lll}
(\rho) & (P \rightarrowtriangle G_2)\, G_3 & \to_\rho\ G_2\ [P \ll G_3] \\
 & (P \rightarrowtriangle G_2)\ [E]\, G_3 & \to_\rho\ G_2\ [P \ll G_3, E] \\
(\delta) & (G_1 \wr G_2)\, G_3 & \to_\delta\ G_1\, G_3 \wr G_2\, G_3 \\
 & (G_1 \wr G_2)\ [E]\, G_3 & \to_\delta\ (G_1\, G_3 \wr G_2\, G_3)\ [E]
\end{array}$$

MATCHING RULES:

$$\begin{array}{lll}
(\mathit{propagate}) & P \ll (G\ [E]) & \to_p\ P \ll G, E \quad \mathit{if}\ P \neq x \\
(\mathit{decompose}) & K(G_1, \ldots, G_n) \ll K(G'_1, \ldots, G'_n) & \to_{dk}\ G_1 \ll G'_1, \ldots, G_n \ll G'_n \\
 & & \quad \mathit{with}\ n \geq 0 \\
(\mathit{solved}) & x \ll G, E & \to_s\ x = G, E \quad \mathit{if}\ x \notin \mathcal{DV}(E)
\end{array}$$

GRAPH RULES:

$$\begin{array}{lll}
(\mathit{external\ sub}) & \mathsf{Ctx}[y]\ [y = G, E] & \to_{es}\ \mathsf{Ctx}[G]\ [y = G, E] \\
(\mathit{acyclic\ sub}) & G\ [P \lll \mathsf{Ctx}[y], y = G_1, E] & \to_{ac}\ G\ [P \lll \mathsf{Ctx}[G_1], y = G_1, E] \\
 & & \quad \mathit{if}\ G_1\ \text{is a variable or}\ (x > y,\ \forall x \in \mathcal{FV}(P)) \\
 & & \quad \mathit{where} \lll \in \{=, \ll\} \\
(\mathit{garbage}) & G\ [E, x = G'] & \to_{gc}\ G\ [E] \\
 & & \quad \mathit{if}\ x \notin \mathcal{FV}(E) \cup \mathcal{FV}(G) \\
 & G\ [\epsilon] & \to_{gc}\ G \\
(\mathit{black\ hole}) & \mathsf{Ctx}[x]\ [x =_\circ x, E] & \to_{bh}\ \mathsf{Ctx}[\bullet]\ [x =_\circ x, E] \\
 & G\ [P \lll \mathsf{Ctx}[y], y =_\circ y, E] & \to_{bh}\ G\ [P \lll \mathsf{Ctx}[\bullet], y =_\circ y, E] \\
 & & \quad \mathit{if}\ \ x > y,\ \forall x \in \mathcal{FV}(P)
\end{array}$$

Fig. 2. Small-step semantics of the ρ_g-calculus

these rules, an additional rule dealing with the presence of constraints is considered.

The MATCHING RULES and in particular the rule *decompose* are strongly related to the theory modulo which we want to compute the solutions of the matching. In this paper we consider the syntactical matching, which is known to be decidable, but extensions to more elaborated theories are possible.

The GRAPH RULES are inherited from the cyclic λ-calculus [AK97]. The first two rules make a copy of a ρ_g-term associated to a recursion variable into a term that is inside the scope of the corresponding constraint. As already mentioned, the substitution rule allows one to make the copies only upwards *w.r.t.* the order defined on the variables of ρ_g-terms. Recall that "_, _" is assumed to be associative, commutative and with ϵ as neutral element, and thus evaluation steps are performed modulo the corresponding theory.

We denote by $\mapsto_{\rho_\mathsf{g}}$ ($\mapsto_\mathcal{M}$) and $\mapsto\!\!\!\twoheadrightarrow_{\rho_\mathsf{g}}$ ($\mapsto\!\!\!\twoheadrightarrow_\mathcal{M}$) the relations induced by the set of rules of Fig. 2 and by the subset of MATCHING RULES, respectively. For any two rules r and s belonging to this set, we will write $\mapsto\!\!\!\twoheadrightarrow_{r,s}$ to express the two steps $\mapsto_r \mapsto_s$.

As mentioned above, the (linear) ρ_g-calculus, with the rewrite relation $\mapsto\!\!\!\twoheadrightarrow_{\rho_\mathsf{g}}$, has been shown to be confluent [BBCK07]. A term G is in *normal form* if no one of the reduction rules of Fig. 2 can be applied to G. A reduction of a term H into its normal form G, when it exists, is denoted by $H \mapsto\!\!\!\twoheadrightarrow_{\rho_\mathsf{g}}^{!} G$.

Example 2.3 [A simple reduction]

$$\begin{array}{l}
\quad (f(a,a) \rightarrowtriangle a)\, (f(y,y)\ [y = a]) \\
\mapsto_\rho\ \ a\ [f(a,a) \ll f(y,y)\ [y = a]] \ \mapsto_p\ \ a\ [f(a,a) \ll f(y,y), y = a] \\
\mapsto_{dk}\ a\ [a \ll y, a \ll y, y = a] \ = \ a\ [a \ll y, y = a] \quad \mathit{(by\ idempotency)} \\
\mapsto_{ac}\ a\ [a \ll a, y = a] \ \mapsto_{dk}\ \ a\ [y = a] \ \mapsto_{gc}\ a
\end{array}$$

Example 2.4 [Reduction to the normal form]

Consider the term $G = f(y, y)\ [y = z\ f(a), z = f(x) \rightarrow x]$. We show one of the possible reductions of G to its normal form.

$$\begin{array}{ll}
 & f(y, y)\ [y = z\ f(a), z = f(x) \rightarrow x] \\
\mapsto_{ac} & f(y, y)\ [y = (f(x) \rightarrow x)\ f(a), z = f(x) \rightarrow x] \\
\mapsto_{p} & f(y, y)\ [y = x\ [f(x) \ll f(a)], z = f(x) \rightarrow x] \\
\mapsto_{dk} & f(y, y)\ [y = x\ [x \ll a], z = f(x) \rightarrow x] \\
\mapsto_{s} & f(y, y)\ [y = x\ [x = a], z = f(x) \rightarrow x] \\
\mapsto_{es} & f(y, y)\ [y = a\ [x = a], z = f(x) \rightarrow x] \\
\mapsto\!\!\!\!\twoheadrightarrow_{gc} & f(y, y)\ [y = a, z = f(x) \rightarrow x] \\
\mapsto\!\!\!\!\twoheadrightarrow_{es} & f(a, a)\ [y = a, z = f(x) \rightarrow x] \mapsto_{gc} f(a, a)
\end{array}$$

Example 2.5 [Encoding of the Peano addition]

We suppose given the constants $0, S, add$ and rec. We define the following ρ-term that computes the addition over Peano integers.

$$plus \ \triangleq\ (rec\, z) \rightarrow \left(\begin{array}{l} (add\, 0\, y) \rightarrow y \\ \wr (add\, (S\, x)\, y) \rightarrow S\ (z\, (rec\, z)\, (add\, x\, y)) \end{array} \right)$$

The variable z will contain a copy of $plus$ to allow "recursive calls". If we use the notations $\overline{m}$, $\overline{m{+}n}$ and $\overline{m{-}n}$ for the terms $S(\ldots(S\,0)\ldots)$ with the right number of S symbols, then the term $plus\,(rec\ plus)\ (add\,\overline{n}\,\overline{m})$ reduces to $\overline{m{+}n}$. Actually, to obtain this result we also need a way of getting rid of some stuck subterms, in which matching definitively fails (see [CLW03,CHW06]).

3 A sharing strategy for ρ_{g}-calculus

In view of a future efficient implementation of the calculus, we are interested in studying suitable strategies that aim at keeping the sharing information as long as possible in ρ_{g}-terms.

Intuitively, the strategy should delay as much as possible the application of the substitution rules, (*external sub*) and (*acyclic sub*), which can break the sharing by duplicating terms. For instance, consider the reduction

$$\begin{array}{ll}
 & f(x, x)\ [x = (a \rightarrow g(b))a] \\
\mapsto_{es} & f((a \rightarrow g(b))a, x)\ [x = (a \rightarrow g(b))a] \\
\mapsto_{es} & f((a \rightarrow g(b))a, (a \rightarrow g(b))a)\ [x = (a \rightarrow g(b))a] \\
\mapsto\!\!\!\!\twoheadrightarrow_{p} & f(g(b)\ [a \ll a], g(b)\ [a \ll a])\ [x = (a \rightarrow g(b))a] \\
\mapsto\!\!\!\!\twoheadrightarrow_{p} & f(g(b)\ [\epsilon], g(b)\ [\epsilon])\ [x = (a \rightarrow g(b))a] \\
\mapsto\!\!\!\!\twoheadrightarrow_{gc} & f(g(b), g(b))
\end{array}$$

The uncontrolled use of susbtitution induces useless and expensive (both in terms of time and space) duplications of terms. For instance, in the case above, the following reduction would be preferable

$$\begin{aligned}
& f(x,x)\ [x = (a \twoheadrightarrow g(b))a] \\
\mapsto_{\rho}\ & f(x,x)\ [x = g(b)\ [a \ll a]] \\
\mapsto_{p}\ & f(x,x)\ [x = g(b)\ [\epsilon]] \\
\mapsto\!\!\!\!\!\to_{gc}\ & f(x,x)\ [x = g(b)]
\end{aligned}$$

The idea underlying the proposed strategy is to constrain substitution rules to be applied only if they are needed for generating new redexes for the basic or matching rules. Note that, in particular, substitutions which do not contribute to generating new basic or matching redexes will never be applied. Hence the strategy will enlarge the class of terms which are in normal form.

For instance, we allow the application of the (*external sub*) rule to the terms $x\ a\ [x = f(x) \twoheadrightarrow x]$ or $x\ a\ [x = a \wr (a \twoheadrightarrow b)]$, since this is useful for creating, respectively, a new (ρ) redex and a new (δ) redex. Instead, (*external sub*) cannot be applied to the terms $f(x,x)\ [x = g(x)]$ or $x\ [x = f(x)]$ which are actually considered in normal form. Note, however, that capturing the notion of "substitution needed for generating a new redex" is not straightforward since more than one substitution step can be needed to generate a new redex for the basic or matching rules as it happens below, where the generated redex is underlined:

$$y\ [y = x\ f(a), x = f(z) \twoheadrightarrow y] \mapsto_{es} x\ f(a)\ [y = x\ f(a), x = f(z) \twoheadrightarrow y]$$

$$\mapsto_{es} \underline{(f(z) \twoheadrightarrow y)\ f(a)}\ [y = x\ f(a), x = f(z) \twoheadrightarrow z]$$

Note that a single step would suffice to generate the redex if we removed the acyclicity constraint for substitutions, allowing the reduction

$$y\ [y = x\ f(a), x = f(z) \twoheadrightarrow y] \mapsto y\ [y = (f(z) \twoheadrightarrow y)\ f(a), x = f(z) \twoheadrightarrow y]$$

The definition of the strategy will rely on the fact, formally proved later, that the above phenomenon is an instance of a completely general case.

There is one more situation in which we want to apply the substitution rules, that is when we have trivial recursion equations of the kind $x = y$ where both sides are single variables, like in $x * y + x\ [x = z, y = z, z = 1]$. In this case, we may want to simplify the term to $(z * z + z)\ [z = 1]$ in which useless names have been eliminated by garbage collection.

Hereafter, we call *basic redex* any term which has one of the shapes $(P \twoheadrightarrow G_2)\ G_3$, $(P \twoheadrightarrow G_2)\ [E]\ G_3$, $(G_1 \wr G_2)\ G_3$ or $(G_1 \wr G_2)\ [E]\ G_3$, which can be reduced using the *Basic rules* in Fig. 2. Similarly, a term of the form $P \ll G$ is called a *matching* redex if it can be reduced by one of the Matching rules.

We define next the reduction strategy we can adopt in the ρ_{g}-calculus to maintain the sharing information during the reduction as long as possible.

Definition 3.1 [Sharing Strategy] The evaluation strategy *SharingStrat* is defined as follows.

(i) All the evaluation rules but (*external sub*) and (*acyclic sub*) are applicable without any restriction.

(ii) The rules (*external sub*) and (*acyclic sub*) are applied to a term G only if no other rule is applicable and if
  (a) their application replaces a variable by a variable (renaming), or
  (b) their application creates (in one step) a basic or a matching redex, or
  (c) the term G has the form $G'\ [x = \mathsf{Ctx}[y], y = \mathsf{Ctx}'[z], E]$, with $x \equiv y$ and $\mathsf{Ctx}[\mathsf{Ctx}'[x]]$ includes a basic or a matching redex.

In other words the rules (*external sub*) and (*acyclic sub*) are applied when their application leads to

- the instantiation of a variable by a variable (condition *(ii)a*);
- the instantiation of an active variable by an abstraction or a structure, which produces a *Basic redex* (condition *(ii)b*);
- the instantiation of a variable in a stuck match equation, which produces a *Matching redex*, i.e., which enables a decomposition or constraint propagation *w.r.t.* the match equation (condition *(ii)b*).

Additionally, condition *(ii)c* captures the fact that, given a term $G\ [E]$ if a cyclic substitution in E would generate a redex, then one is allowed to apply some external substitutions in order to reproduce the same redex in G.

Example 3.2 [Multiplication]

Let us use an infix notation for the constant “$*$”. The following ρ_g-term corresponds to the application of the rewrite rule $\mathcal{R} = x * s(y) \to (x * y + x)$ to the term $1 * s(1)$ where the constant 1 is shared.

$$\begin{array}{ll}
 & (x * s(y) \to (x * y + x))\ (z * s(z)\ [z = 1]) \\
\mapsto_{p} & x * y + x\ [x * s(y) \ll (z * s(z)\ [z = 1])] \\
\mapsto_{p} & x * y + x\ [x * s(y) \ll z * s(z), z = 1] \\
\mapsto\!\!\!\!\to_{dk} & x * y + x\ [x \ll z, y \ll z, z = 1] \\
\mapsto\!\!\!\!\to_{s} & x * y + x\ [x = z, y = z, z = 1] \\
\mapsto\!\!\!\!\to_{es} & (z * z + z)\ [x = z, y = z, z = 1] \quad \textit{(allowed by Def. 3.1(ii) a))} \\
\mapsto\!\!\!\!\to_{gc} & (z * z + z)\ [z = 1]
\end{array}$$

Notice that the term $(z * z + z)\ [z = 1]$ is in normal form *w.r.t.* the strategy *SharingStrat* but can be reduced to $(1 * 1 + 1)$ if no evaluation strategy is used.

Example 3.3 [Reduction to normal form]

We consider the term G of Example 2.4 and we reduce it following the strategy *SharingStrat*. We obtain:

$$\begin{array}{ll}
 & f(y, y)\ [y = z\ f(a), z = f(x) \to x] \\
\mapsto_{ac} & f(y, y)\ [y = (f(x) \to x)\ f(a), z = f(x) \to x] \quad \textit{(by Def. 3.1(ii) b))} \\
\mapsto_{p} & f(y, y)\ [y = x\ [f(x) \ll f(a)], z = f(x) \to x]
\end{array}$$

$\mapsto_{dk}\ f(y,y)\ [y = x\ [x \ll a], z = f(x) \twoheadrightarrow x]$

$\mapsto\!\!\!\!\to_{s}\ f(y,y)\ [y = x\ [x = a], z = f(x) \twoheadrightarrow x]$

$\mapsto_{gc}\ f(y,y)\ [y = x\ [x = a]]$

Note that the normal form with respect to *SharingStrat*, i.e., the term $f(y,y)\ [y = x\ [x = a]]$, represents a graph where the arguments of f are shared. Instead, as shown in Example 2.4, the reduction in the ρ_{g}-calculus with no evaluation strategy leads to the term $f(a,a)$ where the arguments of f are duplicated.

Example 3.4 Consider the ρ_{g}-term $G = f(y)[y = x\ a, x = y \wr b]$. Notice that $x \equiv y$, thus the (*acyclic sub*) rule cannot be applied. We have instead the reduction:

$f(y)[y = x\ a, x = y \wr b] \mapsto_{es} f(x\ a)[y = x\ a, x = y \wr b] \mapsto_{es}$

$f((y \wr b)\ a)[y = x\ a, x = y \wr b] \mapsto_{\delta} f((y\ a \wr b\ a))[y = x\ a, x = y \wr b]$

This derivation is a valid derivation using the strategy *SharingStrat*. Indeed, there exists a cyclic substitution step which transforms the pre-redex $x\ a$ into a basic redex $(y \wr b)\ a$. Hence, the first (*external sub*) rule step can be performed following Def. 3.1(ii) c). The second (*external sub*) rule step is needed to create the basic redex $(y \wr b)\ a$, thus it is allowed for Def. 3.1(ii) b).

4 Properties of the sharing strategy

In this section we will show some basic properties of the ρ_{g}-calculus with the evaluation strategy *SharingStrat*. First, we will show the soundness and completeness of the strategy *SharingStrat w.r.t.* ρ_{g}-calculus (normalising) derivations. In the second part, we will adapt the proof of confluence described in [BBCK07] in order to prove that the ρ_{g}-calculus with the strategy *SharingStrat* is still confluent.

4.1 Soundness and completeness

Here we prove that the reduction strategy proposed for the ρ_{g}-calculus is sound and complete with respect to the one step semantics of the ρ_{g}-calculus as defined in Section 2. Actually, while soundness is immediate, completeness will be proved only for normalising reductions.

Proposition 4.1 (Soundness) *Given two ρ_{g}-terms G and G_n, if $G \mapsto\!\!\!\!\to_{\rho_{\mathsf{g}}} G_n$ in the ρ_{g}-calculus with the strategy* SharingStrat, *then $G \mapsto\!\!\!\!\to_{\rho_{\mathsf{g}}} G_n$ in the ρ_{g}-calculus.*

Proof. Trivial. □

The completeness result relies on a couple of technical lemmata. In the proofs, it will be convenient to refer to a notion of *cyclic substitution*, which consists of the application of the rule *(ac)* without any restriction on the order of variables

$$G\ [P \lll \mathsf{Ctx}[y], y = G_1, E] \to_c G\ [P \lll \mathsf{Ctx}[G_1], y = G_1, E]$$

We will denote by $\mapsto_s$ any application of the substitution rules, *i.e.* (*external sub*) or (*acyclic sub*), possibly cyclic, if this is specified.

We remark that these cyclic substitutions are not allowed in the calculus, but they are just used as a technically convenient tool in proofs. In particular we will use the following simple fact.

Proposition 4.2 *Let G be G' $[x = \mathsf{Ctx}[y], y = \mathsf{Ctx}'[z], E]$ with $\{x, y\} \cap \mathcal{FV}(G') \neq \emptyset$. If $G \mapsto_c G'$ $[x = \mathsf{Ctx}[\mathsf{Ctx}'[z]], y = \mathsf{Ctx}'[z], E]$ and $\mathsf{Ctx}[\mathsf{Ctx}'[z]]$ includes a basic or matching redex, then G is not in normal form with respect to* SharingStrat.

Proof. If $x \equiv y$, then by Definition 3.1*(ii)c* we can apply an external substitution, replacing in G' x or y with their definition. Otherwise, the considered step is a valid application of rule (ac). □

Lemma 4.3 *Let $\mathsf{Ctx}[\_], \mathsf{Ctx}_1[\_]$ be two ρ-contexts such that $\mathsf{Ctx}_1[x]$ is neither a free variable nor a free constrained variable (i.e. neither x nor x $[E]$ with $\mathcal{DV}(E) \cap \{x\} = \emptyset$). Let $\mathsf{Ctx}[\mathsf{Ctx}_1[y]]$ be a ρ-term without redexes and $\mathsf{Ctx}[\mathsf{Ctx}_1[G]]$ be a ρ-term containing a* Basic *or* Matching *redex. Then, $\mathsf{Ctx}_1[G]$ contains a* Basic *or* Matching *redex.*

Proof. [Sketch.] By induction on the form of $\mathsf{Ctx}[\_]$. The interesting cases are the followings:

- $\mathsf{Ctx}[\_]$ *is an application.* In this case $\mathsf{Ctx}[\_] = \mathsf{Ctx}'[\_]\ G'$ or $\mathsf{Ctx}[\_] = G'\ \mathsf{Ctx}'[\_]$. If $\mathsf{Ctx}'[x]$ is not a free (constrained) variable, then we conclude by inductive hypothesis. Hence, either $\mathsf{Ctx}[\_] = \_\ G'$ or $\mathsf{Ctx}[\_] = G'\ \_$.

  If $\mathsf{Ctx}[\_] = \_\ G'$, note that $\mathsf{Ctx}_1[y]$ cannot be a variable and it cannot be an abstraction or a structure (otherwise $\mathsf{Ctx}[\mathsf{Ctx}_1[y]]$ would contain a redex). If $\mathsf{Ctx}_1[y]$ is an application the property clearly holds since no new redexes can be created by instantiation. If $\mathsf{Ctx}_1[y]$ is a constraint application of the form H $[E]$ then, again, H cannot be an abstraction or a structure (otherwise $\mathsf{Ctx}[\mathsf{Ctx}_1[y]]$ would contain a redex). If $H = y$ then it can be instantiated by G and create a new redex only if $y \notin \mathcal{DV}(E)$ but this contradicts the hypothesis.

  If instead, $\mathsf{Ctx}[\_] = G'\ \_$ the property trivially holds.

- $\mathsf{Ctx}[\_]$ is a matching equation. Then $\mathsf{Ctx}[\_] = G' \ll \mathsf{Ctx}'[\_]$, and as above, when $\mathsf{Ctx}'[\_]$ is non-empty we conclude by inductive hypothesis. Thus, let $\mathsf{Ctx}[\_] = \_\ G'$. Since the term $f(G_1, G_2, \ldots, G_n) \ll \mathsf{Ctx}_1[y]$ contains redexes then $\mathsf{Ctx}_1[y]$ is not of the form $f(H_1, H_2, \ldots, H_n)$ or H $[E]$. The term $f(G_1, G_2, \ldots, G_n) \ll \mathsf{Ctx}_1[G]$ is a redex only if $\mathsf{Ctx}_1[G]$ has the form $f(H_1, H_2, \ldots, H_n)$ or H $[E]$ and this is possible only if $\mathsf{Ctx}_1[x]$ is a free (constrained) variable (which contradicts the hypothesis). □

A key point is that it is not possible to create a Basic or Matching redex by further reducing a term that is in normal form *w.r.t.* the reduction strategy. To prove this result, we use the following lemma.

Lemma 4.4 *Let G, G_1, G_2 be ρ_{g}-terms not containing trivial recursion equations, i.e. equations of the form $x = y$. Let $G \mapsto_s G_1 \mapsto_s G_2$ be two (possibly cyclic) substitution steps such that G and G_1 do not contain a* Basic *or* Matching *redex and G_2 does. Then, there exists a (possibly cyclic) substitution step $G \mapsto_s G_2'$, such that the* Basic *or* Matching *redex is present in G_2'.*

Proof.

- Consider the two-steps *external sub* reduction $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{es} \mathsf{Ctx}[\mathsf{Ctx}_1[z]]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{es} \mathsf{Ctx}[\mathsf{Ctx}_1[H]]\ [y = \mathsf{Ctx}_1[z], z = H, E]$ where only the last term contains a BASIC or MATCHING redex. Since by hypothesis $\mathsf{Ctx}_1[\_]$ is not empty, by Lemma 4.3, we know that the redex is in the term $\mathsf{Ctx}_1[H]$. Hence we can build the following one-step reduction: $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{ac} \mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[H], z = H, E]$ Notice that this substitution step may be cyclic, if y and z are cyclically equivalent.
- For the two-steps *acyclic sub* reduction $x\ [x = \mathsf{Ctx}[y], y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{ac} x\ [x = \mathsf{Ctx}[\mathsf{Ctx}_1[z]], y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{es} x\ [x = \mathsf{Ctx}[\mathsf{Ctx}_1[H]], y = \mathsf{Ctx}_1[z], z = H, E]$ we proceed similarly as in the previous case.
- Consider the two-steps reduction $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{ac} \mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[H], z = H, E] \mapsto_{es} \mathsf{Ctx}[\mathsf{Ctx}_1[H]]\ [y = \mathsf{Ctx}_1[H], z = H, E]$ Since by hypothesis $\mathsf{Ctx}_1[\_]$ is not empty, the first *acyclic sub* step instantiates a variable in the term $\mathsf{Ctx}_1[z]$ without changing its structure. Thus, to create a redex, it is sufficient to perform the one-step reduction $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{es} \mathsf{Ctx}[\mathsf{Ctx}_1[z]]\ [y = \mathsf{Ctx}_1[z], z = H, E]$.
- Consider the two-steps reduction $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{es} \mathsf{Ctx}[\mathsf{Ctx}_1[z]]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{ac} \mathsf{Ctx}[\mathsf{Ctx}_1[z]]\ [y = \mathsf{Ctx}_1[H], z = H, E]$ The redex is created in $\mathsf{Ctx}_1[H]$. The first *external sub* step copies a sub-term in the graph but is without effect *w.r.t.* redex creation. We thus can build the one-step reduction $\mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[z], z = H, E] \mapsto_{ac} \mathsf{Ctx}[y]\ [y = \mathsf{Ctx}_1[H], z = H, E]$.

□

Corollary 4.5 *Let G be a ρ_{g}-term with no trivial recursion equations, and let $G \mapsto\!\!\!\to_s G_n$ such that G_n contains a* BASIC *or* MATCHING *redex and G does not, then there exists a a (possibly cyclic) substitution step $G \mapsto_s G'_n$, such that the* BASIC *or* MATCHING *redex is present in G'_n.*

Proof. By induction, using Lemma 4.4. □

Using the above result and exploiting Proposition 4.2 we easily prove the result below, from which compleneteness follows.

Corollary 4.6 *If a ρ_{g}-term G is in normal form with respect to the strategy* SharingStrat *and $G \mapsto\!\!\!\to_{es,ac} G_n$, then G_n is in normal form with respect to the strategy* SharingStrat.

Theorem 4.7 (Completeness) *Given a normalising ρ_{g}-term G, if $G \mapsto\!\!\!\to_{\rho\mathsf{g}}^{!} G_n$ in the ρ_{g}-calculus, then there exists a ρ_{g}-term G_m such that $G \mapsto\!\!\!\to_{\rho\mathsf{g}}^{!} G_m$ in the ρ_{g}-calculus with the strategy* SharingStrat *and $G_m \mapsto\!\!\!\to_{es,ac} G_n$.*

Proof. First, notice that in the ρ_{g}-calculus with the strategy *SharingStrat*, the reduction $G \mapsto\!\!\!\to_{\rho\mathsf{g}} \ldots$ cannot be infinite, otherwise we would have an infinite reduction also in the ρ_{g}-calculus. Thus we have $G \mapsto\!\!\!\to_{\rho\mathsf{g}}^{!} G_m$. Moreover, we have $G_m \mapsto\!\!\!\to_{\rho\mathsf{g}}^{!} G_n$ in the ρ_{g}-calculus, since the calculus is confluent. In order to conclude we have to prove that $G_m \mapsto\!\!\!\to_{es,ac} G_n$ (using only substitution steps).

This follows immediately from Corollary 4.6. In fact, by contradiction, if there

$$\begin{tikzcd}
\cdot \arrow[rr, "R"] \arrow[d, "R"'] & & \cdot \arrow[d, dotted, "R"] \\
\cdot \arrow[r, dotted, two heads, "R"'] & \cdot \arrow[r, phantom, "\sim_{AC1}"] & \cdot
\end{tikzcd}$$

Local confluence

$$\begin{tikzcd}
\cdot \arrow[rr, two heads, "R"] \arrow[d, two heads, "R"'] & & \cdot \arrow[d, dotted, two heads, "R"] \\
\cdot \arrow[r, dotted, two heads, "R"'] & \cdot \arrow[r, phantom, "\sim_{AC1}"] & \cdot
\end{tikzcd}$$

Confluence

$$\begin{tikzcd}
\cdot \arrow[rr, two heads, "R_2"] \arrow[d, two heads, "R_1"'] & & \cdot \arrow[d, dotted, two heads, "R_2"] \\
\cdot \arrow[r, dotted, two heads, "R_1"'] & \cdot \arrow[r, phantom, "\sim_{AC1}"] & \cdot
\end{tikzcd}$$

Commutation

$$\begin{tikzcd}
\cdot \arrow[r, phantom, "\sim_{AC1}"] \arrow[d, "R"'] & \cdot \arrow[d, dotted, "R"] \\
\cdot \arrow[r, phantom, "\sim_{AC1}"] & \cdot
\end{tikzcd}$$

Compatibility

Fig. 3. Properties of rewriting modulo *AC1*

were a reduction $G_n \mapsto\!\!\!\rightarrow_{es,ac} G'_n \mapsto_M G''_n$, then, by Corollary 4.6, G_n would not be in normal form *w.r.t.* the strategy *SharingStrat*. □

Notice that we cannot expect completeness to hold in general, since "useless" unsharing followed by the reduction of some basic or matching redexed cannot be simulated while obeying *SharingStrat*. For instance, the result in Theorem 4.7 would not hold for the derivation

$$\begin{aligned}
& f(x,x)\ [x = (a \twoheadrightarrow b)\ a] \\
& \quad \mapsto_{es} f((a \twoheadrightarrow b)\ a, x)\ [x = (a \twoheadrightarrow b)\ a] \\
& \quad \mapsto\!\!\!\rightarrow_{\rho,dk} f(b,x)\ [x = (a \twoheadrightarrow b)\ a]
\end{aligned}$$

4.2 Confluence

We will next show that the (linear) ρ_{g}-calculus calculus with the evaluation strategy *SharingStrat* is confluent. The proof is obtained by adapting the confluence proof for the ρ_{g}-calculus [BBCK07].

As already mentioned, in the ρ_{g}-calculus, rewriting can be thought of as acting over equivalence classes of ρ_{g}-graphs with respect to the congruence relation, denoted by $\sim_{AC1}$ or simply *AC1*, generated by the associativity, commutativity and neutral element axioms for the "_, _" operator. The relation induced over *AC1*-equivalence classes is written $\mapsto_{\rho_g/AC1}$. Concretely, in the proof, the notion of rewriting modulo *AC1* [PS81], denoted $\mapsto_{\rho_g,AC1}$, is used.

Figure 4.2 provides a graphical representation of some properties of a relation R modulo the congruence relation $\sim_{AC1}$. A formal definition can be found in [Ohl98].

A key property in the confluence proof is the compatibility of the reduction relation with respect to the equivalence on terms, that holds for any subset of rules of the ρ_{g}-calculus. This property ensures that the rewrite relation is particularly well-behaved *w.r.t.* the congruence relation *AC1*.

Clearly, according to the strategy *SharingStrat*, the order in which constraints are listed does not influence the applicability of substitution rules. Therefore compatibility for the ρ_{g}-calculus with strategy *SharingStrat* can be proved exactly as for in the unrestricted calculus.

Lemma 4.8 (Compatibility of ρ_g) *Compatibility with AC1 holds for any rule r of the ρ_{g}-calculus with strategy* SharingStrat.

$$\leftarrow\!\!\dashv_{r,AC1} \cdot \sim_{AC1} \quad \subseteq \quad \sim_{AC1} \cdot \leftarrow\!\!\dashv_{r,AC1}$$

Following the proof in [BBCK07], the evaluation rules of the $\rho_{\mathtt{g}}$-calculus are split into two subsets for which confluence is first proved independently. Then, this intermediate result, together with a commutation lemma, is used for proving the confluence of the union of the two subsets. Here this proof is adapted to the $\rho_{\mathtt{g}}$-calculus calculus with the strategy *SharingStrat*, under the same assumptions of linearity for patterns and constraints.

The first subset of rules, called τ, includes (ρ), (*propagate*), (*decompose*), (*solved*), (*garbage*) and (*black hole*), and the second one, called Σ, consists of the remaining rules, *i.e.* (*external sub*), (*acyclic sub*) and the (δ). Since only the latter set includes rules whose application is constrained by the strategy *SharingStrat*, only the proofs concerning the relation induced by this set should be adapted *w.r.t.* to the unrestricted version of the calculus. In what follows we detail the corresponding proofs, while the properties that can be inherited from the unrestricted calculus [BBCK07] are just stated.

Proposition 4.9 *The relation τ is confluent modulo AC1 under the strategy* SharingStrat*.*

The relation induced by the set of rules Σ is shown to be confluent adapting the *complete development method* defined for the λ-calculus: a terminating version of the relation (the development), denoted $\underline{\Sigma}$, can be defined and its properties are used for deducing the confluence of the original rewrite relation. Due to space constraints, the development relation cannot be defined in full details here. Roughly, a step $G \to_{\underline{\Sigma}} H$ of such relation consists of the complete development, with respect to Σ, of a set of redexes selected in the starting term G. The notation $\underline{\Sigma}$ arises from the fact that the selection is done by underlining the redexes.

First of all, notice that since the strategy *SharingStrat* affects the rewrite relation by restraining the application of the substitution rules, the relation $\underline{\Sigma}$ is clearly still normalising under this strategy. Hence, for proving its confluence, we simply need to verify its local confluence [Ohl98].

Lemma 4.10 *The relation* $(\underline{es \cup ac})$ *is locally confluent modulo AC1 under the strategy* SharingStrat*:*

$$\begin{tikzcd}
G \arrow[r, mapsto, "\underline{es,ac}"'] \arrow[d, mapsto, "\underline{es,ac}"'] & & G_1 \arrow[d, dotted, mapsto, "\underline{es \cup ac}"] \\
G_2 \arrow[r, dotted, mapsto, "\underline{es \cup ac}"'] & G_1' \arrow[r, phantom, "\sim_{AC1}"] & G_2'
\end{tikzcd}$$

Proof. By analysis of the critical pairs. If the terms duplicated by the substitutions are simply variables, then local confluence follows from the corresponding result for the $\rho_{\mathtt{g}}$-calculus. A non trivial critical pair arises when one of the substitution steps occurs in the term duplicated by the other substitution. For example, consider the term $G = G_0\ [y = \underline{x}\ a, x = f(\underline{z}\ a) \wr b, z = a \rightarrow b]$ and the two (ac)-steps leading respectively to $G_1 = G_0\ [y = \underline{x}\ a, x = f((a \rightarrow b)\ a) \wr b, z = a \rightarrow b]$ and $G_2 = G_0\ [y = (f(\underline{z}\ a) \wr b)\ a, x = f(\underline{z}\ a) \wr b, z = a \rightarrow b]$. We can close the critical pair since there exist two reductions $G_1 \mapsto_{ac} G_3$ and $G_2 \mapsto\!\!\!\twoheadrightarrow_{ac} G_3$ such that

$G_3 = G_0\ [y = (f((a \rightarrowtriangle b)\ a) \wr b)\ a, x = f((a \rightarrowtriangle b)\ a) \wr b, z = a \rightarrowtriangle b]$.

□

The same arguments as for the unrestricted version of the calculus can be used to show that the relations $(\underline{es \cup ac})$ and $(\underline{\delta})$ commute. Using this result and the compatibility of the two relations we obtain the confluence of the $\underline{\Sigma}$ relation and then of the Σ relation.

Proposition 4.11 *The relation Σ is confluent modulo AC1 under the strategy* SharingStrat*.*

Once having proved the confluence of the two rewrite relation independently, we prove general confluence of the $(\rho_g, AC1)$ relation by showing the (strong) commutation of the two subsets of rules [Ohl98].

Lemma 4.12 *The relations τ and Σ commute modulo AC1.*

Proof. Since the relations τ and Σ are compatible with *AC1*, it is enough to show strong commutation between the two relations instead of general commutation:

$$\begin{tikzcd}
G \arrow[rr, mapsto, "\tau"'] \arrow[d, "\Sigma"'] & & G_1 \arrow[d, dotted, "\Sigma"', "0/1"] \\
G_2 \arrow[r, dotted, mapsto, "\tau"'] & G'_1 \arrow[r, phantom, "\sim_{AC1}"] & G'_2
\end{tikzcd}$$

If the applied Σ-rule is the (δ) rule, the diagram can be closed as described in [BBCK07]. We are interested here in the cases where the applied Σ-rule is a substitution rule. We proceed by analysing the critical pairs. The diagram can always be closed under the strategy *SharingStrat*, since the τ-rules do not interfere with the creation of basic redexes. For example :

$$\begin{tikzcd}
P \ll (\mathsf{Ctx}[y]\ [y = H, E]) \arrow[r, mapsto, "p"'] \arrow[d, "es"'] & P \ll \mathsf{Ctx}[y], y = H, E \arrow[d, dotted, "ac"'] \\
P \ll (\mathsf{Ctx}[H]\ [y = H, E]) \arrow[r, dotted, mapsto, "p"'] & P \ll \mathsf{Ctx}[H], y = H, E
\end{tikzcd}$$

The basic redex $\mathsf{Ctx}[H]$ can be created before or after the propagation of the list of constraints. We can reason similarly for the application of the other τ-rules, like the (*decompose*) or the (*garbage*) rule (in this case we may have zero Σ steps for closing the diagram). □

Theorem 4.13 (Confluence of ρ_{g}, *AC1*) *The rewrite relation ρ_{g}, AC1 is confluent modulo AC1 under the strategy* SharingStrat*.*

Finally, the main theorem states the confluence of the $\rho_g/AC1$ relation, by deducing it from the confluence of the $(\rho_g, AC1)$ relation.

Theorem 4.14 (Confluence) *The linear ρ_{g}-calculus with the strategy* SharingStrat *is confluent modulo AC1.*

5 Conclusions

In this paper we have proposed a reduction strategy *SharingStrat* for the ρ_{g}-calculus, an extension of the ρ-calculus able to deal with graph like structures. The strategy *SharingStrat* aims at maintaining the sharing information as long as possible in the ρ_{g}-terms and is shown to be sound and complete (for normalising terms). Moreover, the ρ_{g}-calculus with the strategy *SharingStrat* is shown to be confluent, under some restrictions of linearity on patterns.

There are several interesting directions for future research. We intend to investigate the issue of optimality for the reduction strategy, where the notion of "optimal" has to be formally defined, for example in terms of time, space or sharing conservation. In this case a natural reference to compare with would be the work on optimal reduction for lambda calculus. Another matter for future work is to model the rewrite strategy not at the meta level, but in the calculus itself. Taking inspiration from analogous work in the ρ-calculus [CKLW03], we would like to have rewrite rules as primal strategies and iterate rewritings on a ρ_{g}-term adding a fix-point operator to the calculus. Moreover, to detect failures of rewrite rule application at some occurences, we need also to define an exception handling mechanism.

References

[AK96] Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Foundamenta Informaticae*, 26(3-4):207–240, 1996.

[AK97] Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. *Information and Computation*, 139(2):154–233, 1997.

[BBCK05] C. Bertolissi, P. Baldan, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. In M. Fernandez, editor, *2nd International Workshop on Term Graph Rewriting*, volume 127, pages 21–41, Roma, Italy, September 2005. Electronic Notes in Theoretical Computer Science.

[BBCK07] C. Bertolissi, P. Baldan, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. To appear in *Mathematical Structures in Computer Science*, 2007.

[BCKL03] G. Barthe, H. Cirstea, C. Kirchner, and L. Liquori. Pure Patterns Type Systems. In *Proceedings of POPL'03: Principles of Programming Languages, New Orleans, USA*, volume 38, pages 250–261. ACM, 2003.

[Ber05] C. Bertolissi. *The graph rewriting calculus: properties and expressive capabilities.* Thèse de Doctorat d'Université, Institut National Polytechnique de Lorraine, Nancy, France, Octobre 2005.

[BvEG$^+$87] H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In *Proceedings of PARLE'87, Parallel Architectures and Languages Europe*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158, Eindhoven, 1987. Springer-Verlag.

[CG99] A. Corradini and F. Gadducci. Rewriting on cyclic structures: Equivalence of operational and categorical descriptions. *Theoretical Informatics and Applications*, 33:467–493, 1999.

[CHW06] Horatiu Cirstea, Clement Houtmann, and Benjamin Wack. Distributive rho-calculus. In *6th International Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science. Elsevier, 2006. to appear.

[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.

[CKLW03] H. Cirstea, C. Kirchner, L. Liquori, and B. Wack. Rewrite strategies in the rewriting calculus. In Bernhard Gramlich and Salvador Lucas, editors, *Third International Workshop on Reduction Strategies in Rewriting and Programming* , Valencia, Spain, June 2003. Electronic Notes in Theoretical Computer Science.

[CLW03] H. Cirstea, L. Liquori, and B. Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. In Stefano Berardi, Mario Coppo, and Ferruccio Damian, editors, *Types for Proofs and Programs (TYPES)*, volume 3085 of *Lecture Notes in Computer Science*, pages 147–171, Torino (Italy), May 2003.

[Cor93] A. Corradini. Term rewriting in CT_{Σ}. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proceedings of TAPSOFT'93, Theory and Practice of Software Development—4th International Joint Conference CAAP/FASE*, pages 468–484. Springer, Berlin, Heidelberg, 1993.

[Ohl98] E. Ohlebusch. Church-Rosser Theorems for Abstract Reduction Modulo an Equivalence Relation. In T. Nipkow, editor, *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA-98)*, volume 1379 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 1998.

[PJ87] S. Peyton-Jones. *The implementation of functional programming languages*. Prentice Hall, Inc., 1987.

[PS81] G. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28:233–264, 1981.

[SPvE93] M. R. Sleep, M. J. Plasmeijer, and M. C. J. D. van Eekelen, editors. *Term graph rewriting: theory and practice*. Wiley, London, 1993.

[Wac05] B. Wack. *Typage et déduction dans le calcul de réécriture*. Thèse de doctorat, Université Henri Poincaré - Nancy I, October, 7- 2005.