A rewriting calculus for cyclic higher-order term graphs

Paolo Baldan, Clara Bertolissi, Horatiu Cirstea, Claude Kirchner

A rewriting calculus for cyclic higher-order term graphs

Paolo Baldan
Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

Clara Bertolissi, Horatiu Cirstea and Claude Kirchner
LORIA, BP 239, 54506 Vandoeuvre-lès-Nancy Cedex France (`name.surname@loria.fr`*)*

Abstract. Introduced at the end of the nineties, the Rewriting Calculus (ρ-calculus, for short) fully integrates term-rewriting and λ-calculus. The rewrite rules, acting as elaborated abstractions, their application and the obtained structured results are first class objects of the calculus. The evaluation mechanism, generalising beta-reduction, strongly relies on term matching in various theories.

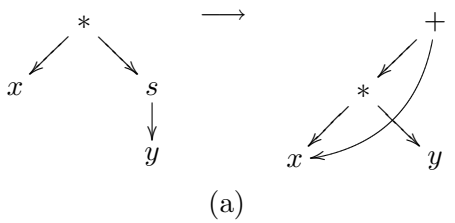
In this paper we propose an extension of the ρ-calculus, called $\rho_{\mathtt{g}}$-calculus, handling structures with cycles and sharing rather than simple terms. This is obtained by using unification constraints in addition to the standard ρ-calculus matching constraints, which leads to a term-graph representation in an equational style. As for the classical ρ-calculus, 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 show that the (linear) $\rho_{\mathtt{g}}$-calculus is confluent. The proof of this result is quite elaborated, due to the non-termination of the system and to the fact that $\rho_{\mathtt{g}}$-calculus-terms are considered modulo an equational theory. We also show that the $\rho_{\mathtt{g}}$-calculus is expressive enough to simulate first-order (equational) term-graph rewriting and λ-calculus with explicit recursion (modelled using a `letrec` like construct).

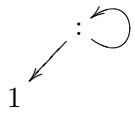
Introduction

Main interests for term rewriting steam 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 analysing 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 (Barendregt et al., 1987). 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 (e.g., when the sharing is maximal, a redex appearing in a shared subterm will be reduced at most once and equality tests can be done in constant time). We can take as example the definition of multiplication by the rewrite system $\mathcal{R} = \{x * 0 \to 0, \quad x * s(y) \to (x * y) + x\}$. If we represent it using graphs, we will write the second rule by duplicating the reference to x instead of duplicating x itself (see Figure 1(a)).

Graph rewriting is a useful technique for the optimisation of functional and declarative languages implementation (Peyton-Jones, 1987). Moreover, the possibility of defining cycles leads to an increased expressive power that allows one to represent easily regular infinite data structures. For example, if ":" denotes the concatenation operator, an infinite list of ones can be modelled as a cyclic list $ones = 1 : ones$, represented by the cyclic graph of Figure 1(b). Cyclic term graph rewriting has been widely studied, both from an operational (Barendregt



(a)



(b)

Figure 1. (a) Rule for multiplication with sharing (b) Cyclic representation of an infinite list of ones.

et al., 1987; Ariola and Klop, 1996) and from a categorical/logical point of view (Corradini and Gadducci, 1999) (see also (Sleep et al., 1993) for a survey on term graph rewriting).

In this context, an abstract model generalising the λ-calculus and adding cycles and sharing features has been proposed in (Ariola and Klop, 1997). Their approach consists of an equational framework that models the λ-calculus extended with explicit recursion. A λ-graph is treated as a system of recursion equations involving λ-terms and rewriting is described as a sequence of equational transformations. This work allows for the combination of graphical structures with the higher-order capabilities of the λ-calculus. A last important ingredient is still missing: pattern matching. The possibility of discriminating using pattern matching could be encoded, in particular in the λ-calculus, but it is much more attractive to directly discriminate and to use indeed rewriting. Programs become quite compact and the encoding of data type structures is no longer necessary.

The rewriting calculus (ρ-calculus, for short) has been introduced in the late nineties as a natural generalisation of term rewriting and of the λ-calculus (Cirstea and Kirchner, 2001). It has been shown to be a very expressive framework e.g. to express object calculi (Cirstea et al., 2001) and it has been equipped with powerful type systems (Barthe et al., 2003). One essential component of the ρ-calculus are the matching constraints that are generated by the generalisation of the β-reduction called ρ-reduction. By making this matching step explicit and the matching constraints first class objects of the calculus, we can allow for an explicit handling of constraints instead of substitutions (Cirstea et al., 2004).

The main contribution of this paper consists of a new system, called the $\rho_{\mathtt{g}}$-calculus, that generalises the cyclic λ-calculus as the standard ρ-calculus generalises the classical λ-calculus.

In the $\rho_{\mathtt{g}}$-calculus any term is associated with a list of constraints consisting of recursion equations, used to express sharing and cycles, and matching constraints, arising from the fact that computations related to the matching are made explicit and performed at the object-level. The order and multiplicity of constraints in a list is inessential and the addition of an empty list of constraints is irrelevant in a $\rho_{\mathtt{g}}$-term. Hence, formally, the conjunction operator which is used to build lists of constraints is assumed to be associative, commutative and idempotent, with the empty list of constraints as neutral element. As a consequence, reductions take place over equivalence classes of terms rather than over single terms and this fact must be considered when reasoning on the rewrite relation induced by the evaluation rules of the calculus.

The calculus is shown to be confluent, under some linearity restrictions on patterns. The proof method generalises the proof of confluence of the cyclic λ-calculus (Ariola and Klop, 1997) to the setting of rewriting modulo an equational theory (Ohlebusch, 1998) and moreover it adapts the proof to deal with terms containing patterns and match equations. More precisely, the concept of "development" and the property of "finiteness of developments", as defined in the theory of the classical λ-calculus (Barendregt, 1984), play a central role in the proof.

The $\rho_{\mathtt{g}}$-calculus is shown to be an expressive formalism that generalises of both the plain ρ-calculus and the λ-calculus extended with explicit recursion, providing an homogeneous framework for pattern matching and higher-order graphical structures. Moreover, we show that (equational) term graph rewriting can be naturally encoded in the $\rho_{\mathtt{g}}$-calculus. More specifically, we prove that matching in the $\rho_{\mathtt{g}}$-calculus is well-behaved *w.r.t.* the notion of homomorphism on term graphs and that any reduction step in a term graph rewrite system can be simulated in the $\rho_{\mathtt{g}}$-calculus.

The paper is organised as follows. In the first section we review the two systems which inspired our new calculus, the standard ρ-calculus (Cirstea and Kirchner, 2001) and the cyclic λ-calculus (Ariola and Klop, 1997), and we briefly describe first-order term graph rewrite systems following an equational approach (Ariola and Klop, 1996). In Section 2 we present the $\rho_{\mathtt{g}}$-calculus with its syntax and its small-step semantics, giving some examples of $\rho_{\mathtt{g}}$-graphs and reductions in the system. In Section 3 we first recall some notions of rewriting in an equational setting in order to show the proof of the confluence of the calculus. After a general

presentation, we provide a detailed proof of the result. In Section 4 we show that the $\rho_{\mathtt{g}}$-calculus is a generalisation of the ρ-calculus and of the cyclic λ-calculus. We show also that first order term graph rewriting reductions can be simulated in the $\rho_{\mathtt{g}}$-calculus. We conclude in Section 5 by presenting some perspectives of future work.

1. General setup

1.1. The rewriting calculus

The ρ-calculus was introduced as a calculus where all the basic ingredients of rewriting are made explicit, in particular the notions of rule abstraction (represented by operator "$\twoheadrightarrow$"), rule application (represented by term juxtaposition) and collection of results (represented by operator "$\wr$"). Depending on the theory behind operator "$\wr$" the results can be grouped together, for example, in lists (when "$\wr$" is associative) or in multi-sets (when "$\wr$" is associative and commutative) or in sets (when "$\wr$" is associative, commutative and idempotent). This operator is useful for representing the (non-deterministic) application of a set of rewrite rules and consequently, the set of possible results. The usual λ-abstraction $\lambda x.t$ is replaced by a rule abstraction $P \twoheadrightarrow T$, where, in the most general case, P is a generic ρ-term. Usually some restrictions are imposed on the shape of P to get desirable properties for the calculus.

The set of ρ-terms is defined as follows:

$$\mathcal{T}, \mathcal{P} \ ::= \ \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}[\mathcal{P} \ll \mathcal{T}] \mid \mathcal{T} \ \mathcal{T} \mid \mathcal{T} \wr \mathcal{T}$$

The symbols $T, U, L, R, \ldots$ range over the set $\mathcal{T}$ of terms, the symbols $x, y, z, \ldots$ range over the set $\mathcal{X}$ of variables, the symbols $a, b, c, \ldots, f, g, h$ range over a set $\mathcal{K}$ of constants. We assume that the (hidden) application operator $(\_\ \_)$ associates to the left, while the other operators associate to the right. The priority of the application operator is higher than that of $\_[\_ \ll \_]$ which is higher than that of $\_ \twoheadrightarrow \_$ which is, in turn, of higher priority than $\_ \wr \_$. Terms of the form $(T_0\ T_1 \cdots T_n)$ will be often denoted $T_0(T_1, \ldots, T_n)$. Given a term T, a *position* ω in T is a sequence $f_1 i_1 f_2 i_2 \ldots f_n i_n$, such that $T = f_1(T_1, \ldots, T_{i_1}, \ldots)$ and $f_2 i_2 \ldots f_n i_n$ is a position in T_{i_1}. We call ϵ the empty sequence denoting the head position of t. A sub-term of T at position ω in T is denoted $T_{|\omega}$. We will use the notation $T_{\lceil U \rceil_\omega}$ to specify that T has a sub-term U at position ω and the notation $T_{[U]_\omega}$ to denote the term obtained from T by replacing the sub-term $T_{|\omega}$ by U.

We define next the set of free variables of a ρ-term, generalising the notion of free variables of the λ-calculus.

DEFINITION 1 (Free and active variables). *The set of* free *variables is defined by:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{FV}(f) & \triangleq & \{\,\} & \mathcal{FV}(T_1\ T_2) & \triangleq & \mathcal{FV}(T_1) \cup \mathcal{FV}(T_2) \\
\mathcal{FV}(X) & \triangleq & \{X\} & \mathcal{FV}(T_1 \wr T_2) & \triangleq & \mathcal{FV}(T_1) \cup \mathcal{FV}(T_2) \\
\mathcal{FV}(P \twoheadrightarrow T) & \triangleq & \mathcal{FV}(T) \setminus \mathcal{FV}(P) & \mathcal{FV}(T_1[P \ll T_2]) & \triangleq & (\mathcal{FV}(T_2) \setminus \mathcal{FV}(P)) \cup \mathcal{FV}(T_1)
\end{array}$$

A term is called closed *if it has no free variables. A variable is* active *in a term T when it appears free in the left-hand side of an application occurring in T.*

In an abstraction $P \twoheadrightarrow T$, the free variables of P bind the corresponding variables in T, while in $T_2[P \ll T_1]$, the free variables of P are bound in T_2 (but not in T_1).

As it commonly happens in calculi involving binders, we work modulo the *α-convention* (Church, 1941), *i.e.* two terms that differ only for the names of their bound variables are considered α-equivalent, and modulo the *hygiene-convention* of (Barendregt, 1984), *i.e.* free and bound variables are assumed to have different names.

$$\begin{array}{lrcl}
(\rho) & (P \twoheadrightarrow T)U & \mapsto_{\rho} & T[P \ll U] \\
(\sigma) & T[P \ll U] & \mapsto_{\sigma} & \sigma_{P \ll U}(T) \\
(\delta) & (T_1 \wr T_2)\ T_3 & \mapsto_{\delta} & T_1\ T_3 \wr T_2\ T_3
\end{array}$$

Figure 2. Small-step semantics of ρ-calculus.

Since we work on equivalence classes induced by the α-conversion, the application of a substitution σ to a term T, denoted by $\sigma(T)$ or $T\sigma$, is defined, as usual, to avoid variable captures.

The small-step reduction semantics is defined by the evaluation rules presented in Figure 2. The application of a rewrite rule (abstraction) to a term evaluates via the rule (ρ) to the application of the corresponding constraint to the right-hand side of the rewrite rule. Such a construction is called a *delayed matching constraint*. By rule (σ), if the matching problem between P and U admits a solution σ then the delayed matching constraint evaluates to $\sigma(T)$. The matching power of the ρ-calculus can be regulated using arbitrary theories. Here we consider the ρ-calculus with the empty theory (*i.e.* syntactic matching) that is decidable and has a unique solution. The rule (δ) distributes the application of structures.

Starting from these top-level rules we define, as usually, the context closure denoted $\mapsto_{\rho\sigma\delta}$. The many-step evaluation $\mapsto_{\rho\sigma\delta}^{*}$ is defined as the reflexive-transitive closure of $\mapsto_{\rho\sigma\delta}$.

EXAMPLE 2. *Let $head(cons(x, y)) \twoheadrightarrow x$ be the ρ-abstraction returning the head of a list. If we apply it to $head(cons(a, b))$ we obtain the following reduction:*

$$\begin{array}{rcl}
(head(cons(x,y)) \twoheadrightarrow x)\ head(cons(a,b)) & \mapsto_{\rho} & x[head(cons(x,y)) \ll head(cons(a,b))] \\
& \mapsto_{\sigma} & x\{x/a, y/b\} \\
& = & a
\end{array}$$

1.2. The cyclic lambda calculus

The cyclic λ-calculus introduced in (Ariola and Klop, 1997) generalises the ordinary λ-calculus by allowing to represent sharing and cycles in the λ-calculus terms. This is obtained by adding to the λ-calculus a `letrec`-like construct, in a such way that the new terms, called λ-graphs, are essentially systems of (possibly nested) recursion equations on standard λ-terms. If the system is used without restrictions on the rules, the confluence is lost. The authors restore it by controlling the operations on the recursion equations. The resulting calculus, called $\lambda\phi$, is powerful enough to incorporate the classical λ-calculus (Barendregt, 1984), the $\lambda\mu$-calculus (Parigot, 1992) and the $\lambda\sigma$-calculus with names (Abadi et al., 1991) extended with horizontal and vertical sharing respectively. The syntax of $\lambda\phi$ is the following:

$$t \ ::= \ x \mid f(t_1, \ldots t_n) \mid t_0\ t_1 \mid \lambda x.t \mid \langle t_0 \mid \ x_1 = t_1, \ldots, x_n = t_n \rangle$$

The set of $\lambda\phi$-terms consists of the ordinary λ-terms (*i.e.* variables, functions of fixed arity, applications, abstractions) and of new terms built using a `letrec` construct $\langle t_0 \mid \ x_1 = t_1, \ldots, x_n = t_n \rangle$, where the recursion variables x_i are assumed all distinct, for $i = 1, \ldots, n$. Variables are bound either by lambda abstractions or by recursion equations. Let E denote any unordered sequence of equations $x_1 = t_1, \ldots, x_n = t_n$ and let ϵ be the empty sequence. The notation $x =_{\circ} x$ is an abbreviation for the sequence of recursion equations $x = x_1, \ldots, x_n = x$. Terms are denoted by the symbols $t, s, \ldots$, variables are denoted by the symbols $x, y, z, \ldots$ and constants by the symbols $a, b, c, \ldots, f, g, h$.

A context $\mathsf{Ctx}\{\Box\}$ is a term with a single hole $\Box$ in place of a subterm. Filling the context $\mathsf{Ctx}\{\Box\}$ with a term t yields the term $\mathsf{Ctx}\{t\}$. We denote by $\leq$ the least pre-order on recursion variables such that $x \geq y$ if $x = \mathsf{Ctx}\{y\}$, for some context $\mathsf{Ctx}\{\Box\}$. We write $x > y$ if $x \geq y$ and

(β)	$(\lambda x.t_1)\ t_2$	$\to_\beta$	$\langle t_1 \mid x = t_2\rangle$
(*external sub*)	$\langle \mathsf{Ctx}\{y\} \mid y = t, E\rangle$	$\to_{es}$	$\langle \mathsf{Ctx}\{t\} \mid y = t, E\rangle$
(*acyclic sub*)	$\langle t_1 \mid y = \mathsf{Ctx}\{x\}, x = t_2, E\rangle$	$\to_{ac}$	$\langle t_1 \mid y = \mathsf{Ctx}\{t_2\}, x = t_2, E\rangle$ *if* $y > x$
(*black hole*)	$\langle \mathsf{Ctx}\{x\} \mid x =_\circ x, E\rangle$	$\to_\bullet$	$\langle \mathsf{Ctx}\{\bullet\} \mid x =_\circ x, E\rangle$
	$\langle t \mid y = \mathsf{Ctx}\{x\}, x =_\circ x, E\rangle$	$\to_\bullet$	$\langle t \mid y = \mathsf{Ctx}\{\bullet\}, x =_\circ x, E\rangle$ *if* $y > x$
(*garbage collect*)	$\langle t \mid E, E'\rangle$	$\to_{gc}$	$\langle t \mid E\rangle$ *if* $E' \neq \epsilon$ *and* $E' \perp (E, t)$
	$\langle t \mid \epsilon\rangle$	$\to_{gc}$	t

Figure 3. Evaluation rules of the $\lambda\phi_0$-calculus.

$x \not\equiv y$, where $\equiv$ is the equivalence induced by the pre-order, *i.e.* $x \equiv y$ if $x \geq y \geq x$ (intuitively, if variables x and y occur in a cycle). We write $E \perp (E', t)$ and say that E is *orthogonal* to a sequence of equations E' and a term t, if the recursion variables of E do not intersect the free variables of E' and t. The reduction rules of the basic $\lambda\phi$-calculus, referred to as $\lambda\phi_0$-calculus, are given in Figure 3. Some extensions of this basic set of rules are considered in (Ariola and Klop, 1997), which add either distribution rules ($\lambda\phi_1$) or merging and elimination rules ($\lambda\phi_2$) for the $\langle \_ \mid \_ \rangle$ construct. In the following we will concentrate our attention on the basic system in Figure 3. In the β-rule, the variable x bound by λ becomes bound by a recursion equation after the reduction. The two substitution rules are used to make a copy of a λ-graph associated to a recursion variable. The restriction on the order of recursion variables is introduced to ensure confluence in the case of cyclic configurations of lambda redexes. The condition $E' \neq \epsilon$ in the rule *garbage collect* rule avoids trivial non-terminating reductions.

We denote by $\mapsto_{\lambda\phi}$ the rewrite relation induced by the set of rules of Figure 3 and by $\mapsto\!\!\!\!\to_{\lambda\phi}$ its reflexive and transitive closure.

EXAMPLE 3. *Consider the λ-graph $t = \langle y \mid y = plus(z\ 0, z\ 1), z = \lambda x.s(x)\rangle$ where 0 and 1 are constants and s is meant to represent the successor function. We have the following reduction, where at each step the considered redex is underlined.*

$$
\begin{aligned}
& \langle y \mid y = plus(\underline{z}\ 0, z\ 1), \underline{z = \lambda x.s(x)}\rangle \\
\mapsto_{ac}\ & \langle y \mid y = plus(\underline{\lambda x.s(x)\ 0}, z\ 1), z = \lambda x.s(x)\rangle \\
\mapsto_{\beta}\ & \langle y \mid y = plus(\langle \underline{x} \mid \underline{x = s(0)}\rangle, z\ 1), z = \lambda x.s(x)\rangle \\
\mapsto_{es}\ & \langle y \mid y = plus(\langle s(0) \mid \underline{x = s(0)}\rangle, z\ 1), z = \lambda x.s(x)\rangle \\
\mapsto\!\!\!\!\to_{gc}\ & \langle y \mid y = plus(s(0), \underline{z}\ 1), \underline{z = \lambda x.s(x)}\rangle \\
\mapsto_{ac}\ & \langle y \mid y = plus(s(0), \underline{\lambda x.s(x)\ 1}), z = \lambda x.s(x)\rangle \\
\mapsto\!\!\!\!\to_{\lambda\phi}\ & \langle y \mid y = plus(s(0), s(1)), \underline{z = \lambda x.s(x)}\rangle \\
\mapsto_{gc}\ & \langle y \mid y = plus(s(0), s(1))\rangle
\end{aligned}
$$

1.3. Term graph rewriting

Several presentations have been proposed for term graph rewriting (TGR) (see (Sleep et al., 1993) for a survey). Here we consider an equational presentation in the style of (Ariola and Klop, 1996). Given a set of variables $\mathcal{X}$ and a first-order signature $\mathcal{F}$ with symbols of fixed arity, a term graph over $\mathcal{X}$ and $\mathcal{F}$ is a system of equations of the form $G = \{x_1 \mid x_1 = t_1, \ldots, x_n = t_n\}$ where $t_1, \ldots, t_n$ are terms over $\mathcal{X}$ and $\mathcal{F}$ and the recursion variables x_i are pairwise distinct, for $i = 1, \ldots, n$. The variable x_1 on the left represents the root of the term graph. We call the list of equations the *body* of the term graph and we denote it by E_G, or simply E, when G is clear from the context. The empty list is denoted by ϵ. The variables $x_1, \ldots, x_n$ are bound in

the term graph by the associated recursion equation. The other variables occurring in the term graph G are called *free* and the set of free variables is denoted by $\mathcal{FV}(G)$. A term graph without free variables is called *closed*. We denote the collection of variables appearing in G by $\mathcal{V}ar(G)$. Two α-equivalent term graphs, *i.e.* two term graphs which differ only for the name of bound variables, are considered equal. Cycles may appear in the system and degenerated cycles, *i.e.* equations of the form $x = x$, are replaced by $x = \bullet$ (black hole). A term graph is said to be in *flat form* if all its recursion equations are of the form $x = f(x_1, \ldots, x_n)$, where the variables $x, x_1, \ldots, x_n$ are not necessarily distinct from each other. In the following we will consider only term graphs in flat form and without useless equations (garbage) that we remove automatically during rewriting. A term graph in flat form can be easily interpreted and depicted as a graph taking the set of variables as nodes. We will use the graphical interpretation to help the intuition in the examples.

Rewriting is done by means of term graph rewriting rules.

DEFINITION 4 (Term graph rewrite rule). *A term graph rewrite rule is a pair of term graphs (L, R) such that L and R have the same root, L is not a single variable and $\mathcal{FV}(R) \subseteq \mathcal{FV}(L)$. We say that a rewrite rule is* left-linear *if L is acyclic and each variable appears at most once in the right-hand side of the recursion equations of L.*

In the sequel we will restrict to left-linear rewrite rules.

DEFINITION 5 (Term graph rewrite system). *A* term graph rewrite system *(TGR) consists of a pair $TGR = (\Sigma, \mathcal{R})$ where Σ is a signature and $\mathcal{R}$ is a set of rewrite rules over this signature.*

A rewrite rule can be applied to a term graph if there exists a match between its left-hand side and the graph. We point out that, since matching is often formalised as a possibly non-injective homomorphism from the left-hand side of the rule into the term graph, a rule can match term graphs containing more sharing than its left-hand side. Notice that in the case of term graphs in flat form, considered here, the homomorphism σ is simply a variable (possibly non-injective) renaming.

DEFINITION 6 (Substitution, Matching and Redex).

- *A substitution $\sigma = \{x_1/y_1, \ldots, x_n/y_n\}$ is a map from variables to variables. Its application to a term graph G, denoted $\sigma(G)$, is inductively defined as follows:*

$$\sigma(\{z_1 \mid z_1 = t_1, \ldots, z_n = t_n\}) \triangleq \{\sigma(z_1) \mid \sigma(z_1) = \sigma(t_1), \ldots, \sigma(z_n) = \sigma(t_n)\}$$

$$\sigma(z_i) \triangleq \begin{cases} y_i & \text{if } z_i = x_i \in \{x_1, \ldots, x_n\} \\ z_i & \text{otherwise} \end{cases} \qquad \sigma(f(t_1, \ldots, t_n)) \triangleq f(\sigma(t_1), \ldots, \sigma(t_n))$$

- *A homomorphism (matching) from a term graph L to a term graph G is a substitution σ such that $\sigma(L) \subseteq G$, where the inclusion means that all recursion equations of $\sigma(L)$ are in G, i.e. if $\sigma(L) = \{x_1 \mid E\}$ then $G = \{x_1' \mid E, E'\}$.*

- *A redex in a term graph G is a pair $((L, R), \sigma)$ where (L, R) is a rule and σ is a homomorphism from the left-hand side L of the rule to G. If x is the root of L, we call $\sigma(x)$ the head of the redex.*

We introduce next the notions of path and position, later used to define a rewrite step.

DEFINITION 7 (Path, position). *A* path *in a closed term graph G is a sequence of function symbols interleaved by integers $p = f_1 i_1 f_2 \ldots i_{n-1} f_n$ such that f_{j+1} is the i_j-th argument of f_j, for all $j = 0, \ldots, n$. The sequence of integers $i_1, \ldots, i_{n-1}$ is called the* position *of the node labelled f_n and still denoted with the letter p.*

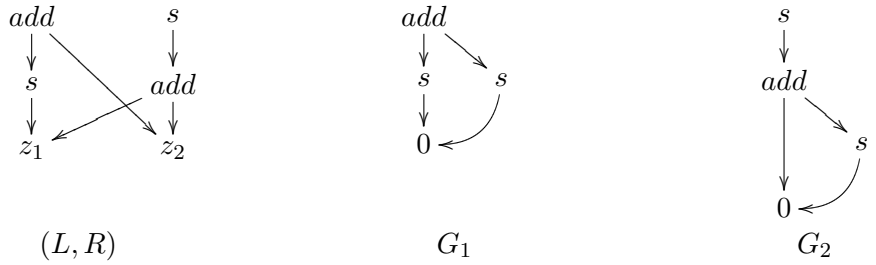


Figure 4. Examples of term graphs.

Similarly to what we have done for terms, we introduce the notations $G_{|p}$ for the subgraph of G at the position p in G, while $G_{\lceil G' \rceil_p}$ specifies that G contains a term graph G' at the position p. In the same situation, if z is the root of G' and $z = t$ is the corresponding equation we will also write $G_{\lceil z=t \rceil_p}$. By the notation $G_{[G']_p}$ we denote the term graph G where the subgraph $G_{|p}$, or more precisely the equation defining the root of $G_{|p}$, has been replaced by G'. Given for instance the two term graphs $G_{\lceil z=t \rceil_p}$ and $G' = z\ [E_{G'}]$, the term graph $G_{[G']_p}$ is obtained from G by replacing the equation $z = t$ by $E_{G'}$, and then possibly performing garbage collection. Intuitively, the term graph G' is attached to the node z in G.

The notions of path and position are used to define a rewrite step.

DEFINITION 8 (Rewrite step). *Let $((L, R), \sigma)$ be a redex occurring in G at the position p. A rewrite step which reduces the redex above consists of removing the equation specified by the head of the redex and of replacing it by the body of $\sigma(R)$, with a fresh choice of bound variables. Using a context notation: $G_{\lceil \sigma(x)=t \rceil_p} \to G_{[\sigma(R)]_p}$*

We give next an example of rewriting. Note that only the root equation of the match gets rewritten and it is replaced by several equations. Renaming is needed to avoid collisions with other variables already in the term graph.

EXAMPLE 9 (Rewriting). *Let $G_1 = \{x_1 \mid x_1 = add(x_2, x_3), x_2 = s(x_4), x_3 = s(x_4), x_4 = 0\}$ be a closed term graph in flat form and let $(L, R) = (\{y_1 \mid y_1 = add(y_2, z_2), y_2 = s(z_1)\}, \{y_1 \mid y_1 = s(y_2), y_2 = add(z_1, z_2)\})$ be a rewrite rule (see Figure 4). Note that in the rule the bound variables are y_1 and y_2, while the free variables are z_1 and z_2. A matching of L in G_1 is given by the substitution $\sigma = \{y_1/x_1, y_2/x_2, z_1/x_4, z_2/x_3\}$. The rewrite step is performed at the root of G_1. We have*

$$\begin{aligned} G_1 &= \{x_1 \mid \underline{x_1 = add(x_2, x_3)}, x_2 = s(x_4), x_3 = s(x_4), x_4 = 0\} \\ &\to \{x_1 \mid \underline{x_1 = s(x_2'), x_2' = add(x_4, x_3)}, x_2 = s(x_4), x_3 = s(x_4), x_4 = 0\} = G_2 \end{aligned}$$

where the underlined equation in G_1 is rewritten into the underlined equations in G_2. The resulting term graph G_2 is depicted in Figure 4.

2. The graph rewriting calculus

2.1. The syntax of $\rho_{\mathtt{g}}$-calculus

The syntax of the $\rho_{\mathtt{g}}$-calculus presented in Figure 5 extends the syntax of the standard ρ-calculus and of the $\rho_{\mathtt{X}}$-calculus (Cirstea et al., 2004), *i.e.* the ρ-calculus with explicit matching and substitution application. As in the plain ρ-calculus, λ-abstraction is generalised by a rule abstraction $P \twoheadrightarrow G$, where P is in general an arbitrary term, referred to as *pattern*. There are two different

Terms

$$
\begin{array}{rcll}
\mathcal{G}, \mathcal{P} & ::= & \mathcal{X} & \text{(Variables)} \\
& | & \mathcal{K} & \text{(Constants)} \\
& | & \mathcal{P} \twoheadrightarrow \mathcal{G} & \text{(Abstraction)} \\
& | & \mathcal{G}\ \mathcal{G} & \text{(Functional application)} \\
& | & \mathcal{G} \wr \mathcal{G} & \text{(Structure)} \\
& | & \mathcal{G}\ [\mathcal{C}] & \text{(Constraint application)}
\end{array}
$$

Constraints

$$
\begin{array}{rcll}
\mathcal{C} & ::= & \epsilon & \text{(Empty constraint)} \\
& | & \mathcal{X} = \mathcal{G} & \text{(Recursion equation)} \\
& | & \mathcal{P} \ll \mathcal{G} & \text{(Match equation)} \\
& | & \mathcal{C}, \mathcal{C} & \text{(Conjunction of constraints)}
\end{array}
$$

Figure 5. Syntax of the ρ_{g}-calculus

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 "$\_ \wr \_$".

As the $\rho_{\mathtt{X}}$-calculus, the ρ_{g}-calculus deals explicitly with matching constraints of the form $P \ll G$ but it introduces also a new kind of constraint, the recursion equations. A recursion equation is a constraint of the form $X = G$ and can be seen as a delayed substitution, or as an environment associated to a term. In the ρ_{g}-calculus constraints are conjunctions (built using the operator "$\_, \_$") of match equations and recursion equations. The empty constraint is denoted by ϵ. The operator "$\_, \_$" is assumed to be associative, commutative and idempotent, 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: "$\_\ \_$"(application), "$\_ \twoheadrightarrow \_$", "$\_ \wr \_$", "$\_\ [\_]$" , "$\_ \ll \_$", "$\_ = \_$" and "$\_, \_$".

The symbols $G, H, \ldots$ range over the set $\mathcal{G}$ of ρ_{g}-graphs, $x, y, z, \ldots$ range over the set $\mathcal{X}$ of variables, $a, b, c, \ldots, f, g, h$ range over a set $\mathcal{K}$ of constants. The symbols $E, F, \ldots$ range over the set $\mathcal{C}$ of constraints.

We call a ρ_{g}-graph *well-formed* if each variable occurs at most once as left-hand side of a recursion equation. All the ρ_{g}-graphs considered in the sequel will be implicitly well-formed.

We call *algebraic* the ρ_{g}-graphs defined by the following grammar:

$$\mathcal{A} \ ::= \ \mathcal{X} \ | \ \mathcal{K} \ | \ (((\mathcal{K}\ \mathcal{A})\ \mathcal{A}) \ldots)\ \mathcal{A} \ | \ \mathcal{A}\ [\mathcal{X} = \mathcal{A}, \ldots, \mathcal{X} = \mathcal{A}]$$

An algebraic term of the form $(((f\ G_1)\ G_2) \ldots)\ G_n$ will be usually written as $f(G_1, G_2, \ldots, G_n)$. We use the symbol $\mathsf{Ctx}\{\Box\}$ for a context with exactly one hole $\Box$ and $\mathsf{Ctx}\{G\}$ for the ρ_{g}-graph obtained from by filling such a hole with G, defined in the obvious way.

DEFINITION 10 (Order, cycle). *We denote by $\leq$ the least pre-order on recursion variables such that $x \geq y$ if $\mathsf{Ctx}_1\{x\} \lll \mathsf{Ctx}_2\{y\}$ for some contexts $\mathsf{Ctx}_i\{\Box\}$, $i = 1, 2$, where the symbol $\lll$ can be the recursion operator $=$ or the match operator $\ll$. The equivalence induced by the pre-order is denoted $\equiv$ and we say that x and y are cyclically equivalent ($x \equiv y$) if $x \geq y \geq x$. We write $x > y$ if $x \geq y$ and $x \not\equiv y$.*

We say that a ρ_{g}-graph is acyclic *if $\geq$ is a partial order (and thus $\equiv$ is the identity on variables).*

Observe that a cyclic ρ_{g}-graph will contain a *cycle*, i.e., a 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 $n, m \in \mathbb{N}$, where $x_0 \equiv x_1 \equiv \ldots \equiv x_n$.

We denote by $\bullet$ (black hole) a constant, already introduced by Ariola and Klop (Ariola and Klop, 1996) using the equational approach and also by Corradini (Corradini, 1993) using the categorical approach, to name "undefined" ρ_{g}-graphs corresponding to the expression $x\ [x = x]$ (self-loop). The notation $x =_\circ x$ is again an abbreviation for the sequence $x = x_1, \ldots, x_n = x$.

The notions of free and bound variables of ρ_{g}-graphs take into account the three binders of the calculus: abstraction, recursion and match. Intuitively, variables which occur free in the left hand-side of any of these operators bound the occurrences of the same variable in the right-hand side of the operator.

Given a constraint $\mathcal{C}$ we will also 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 matching equation $P \ll G$ in $\mathcal{C}$, the set of free variables of P.

DEFINITION 11 (Free, bound, and defined variables). *Given a ρ_{g}-graph G, its free variables, denoted $\mathcal{FV}(G)$, and its bound variables, denoted $\mathcal{BV}(G)$, are recursively defined below:*

G	$\mathcal{BV}(G)$	$\mathcal{FV}(G)$
x	$\emptyset$	$\{x\}$
k	$\emptyset$	$\emptyset$
$G_1\ G_2$	$\mathcal{BV}(G_1) \cup \mathcal{BV}(G_2)$	$\mathcal{FV}(G_1) \cup \mathcal{FV}(G_2)$
$G_1 \wr G_2$	$\mathcal{BV}(G_1) \cup \mathcal{BV}(G_2)$	$\mathcal{FV}(G_1) \cup \mathcal{FV}(G_2)$
$G_1 \twoheadrightarrow G_2$	$\mathcal{FV}(G_1) \cup \mathcal{BV}(G_1) \cup \mathcal{BV}(G_2)$	$\mathcal{FV}(G_2) \setminus \mathcal{FV}(G_1)$
$G\ [\mathcal{C}]$	$\mathcal{BV}(G) \cup \mathcal{BV}(\mathcal{C})$	$(\mathcal{FV}(G) \cup \mathcal{FV}(\mathcal{C})) \setminus \mathcal{DV}(\mathcal{C})$

For a given constraint C, the free variables, denoted $\mathcal{FV}(C)$, the bound variables, denoted $\mathcal{BV}(C)$, and the defined variables, denoted $\mathcal{DV}(C)$, are defined as follows:

C	$\mathcal{BV}(C)$	$\mathcal{FV}(C)$	$\mathcal{DV}(C)$
ϵ	$\emptyset$	$\emptyset$	$\emptyset$
$x = G$	$x \cup \mathcal{BV}(G)$	$\mathcal{FV}(G)$	$\{x\}$
$G_1 \ll G_2$	$\mathcal{FV}(G_1) \cup \mathcal{BV}(G_1) \cup \mathcal{BV}(G_2)$	$\mathcal{FV}(G_2)$	$\mathcal{FV}(G_1)$
C_1, C_2	$\mathcal{BV}(C_1) \cup \mathcal{BV}(C_2)$	$\mathcal{FV}(C_1) \cup \mathcal{FV}(C_2)$	$\mathcal{DV}(C_1) \cup \mathcal{DV}(C_2)$

It is worth remarking that the set of bound variables in the subterm G of a constraint application $G\ [E]$ is the domain of E plus the bound variables of G. For example, the bound variables of the term $(f(y) \twoheadrightarrow y)\ (g(x,z)\ [x \ll f(a)])$ are the variables x and y. Note also that the visibility of a recursion variable is limited to the ρ_{g}-graphs appearing in the list of constraints where the recursion variable is defined and the ρ_{g}-graph to which this list is applied. For example, in the term $f(x,y)\ [x = g(y)\ [y = a]]$ the variable y defined in the recursion equation binds its occurrence in $g(y)$ but not in $f(x,y)$. To avoid confusion and guarantee that free and bound variables have always different names in a ρ_{g}-graph, we assume to work modulo *α-conversion* and to use Barendregt's *"hygiene-convention"*. Using α-conversion, the previous term becomes $f(x,z)\ [x = g(y)\ [y = a]]$ where it is clear that the variable z is free. The operation of α-conversion is used also for defining a capture-free substitution operation over ρ_{g}-graphs.

Since sometimes the notion of free (and bound) variables is not very intuitive, due to the presence of different binders in the calculus and to the fact that the sets of variables of the different constraints in a list are not necessarily disjoint, we give some examples about the visibility of bound variables and the need of renaming variables.

EXAMPLE 12 (Free and bound variables should not have the same name).
Given the ρ_{g}-graph $z\ [z = x \twoheadrightarrow y, y = x + x]$, one may think to naively replace the variable y by $x + x$ in the right-hand side of the abstraction, leading to a variable capture.

This could happen because the previous term does not respect our naming conventions: the variable capture is no longer possible if we consider the legal ρ_{g}-graph $z\ [z = x_1 \twoheadrightarrow y, y = x + x]$ obtained after α-conversion. In order to have the occurrences of the variable x appearing in the second constraint bounded by the arrow, we should use a nested constraint as in the ρ_{g}-graph $z\ [z = x \twoheadrightarrow (y\ [y = x + x])]$.

EXAMPLE 13 (Different bound variables should have different names).
Intuitively, according to the notions of free and bound variable, in a term there cannot be any

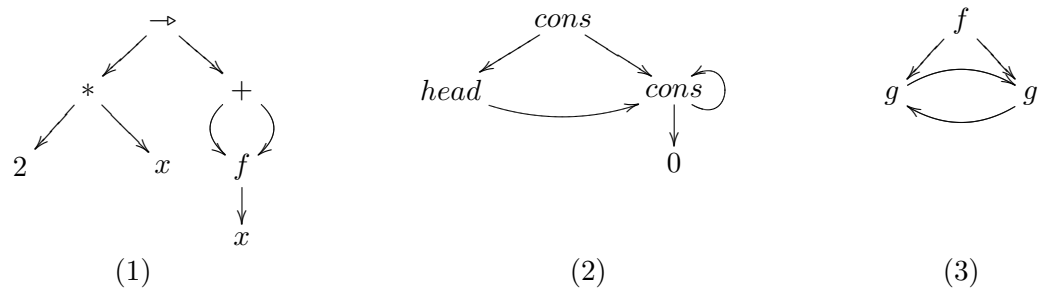


Figure 6. Some $\rho_{\mathtt{g}}$-graphs.

sharing between the left-hand side of rewrite rules and the rest of a $\rho_{\mathtt{g}}$-graph. In other words, the left-hand side of a rewrite rule is self-contained. Sharing inside the left-hand side is allowed and no restrictions are imposed on the right-hand side.

For example, in $f(y, y \rightharpoonup g(y))\ [y = x]$ the first occurrence of y is bound by the recursion variable, while the scope of the y in the abstraction "$\_ \rightharpoonup \_$" is limited to the right-hand side of the abstraction itself. The $\rho_{\mathtt{g}}$-graph should be in fact written (by α-conversion) as $f(y, z \rightharpoonup g(z))\ [y = x]$.

Notice also that it is not possible to express sharing between the left and right hand sides of an abstraction. E.g., in the term $(x \rightharpoonup x)\ [x = a]$ the variable x in the left-hand side of the asbtraction is bound by the x in the right-hand side and thus the term can be α-converted to $(z \rightharpoonup z)\ [x = a]$ which verifies the naming convention.

This naming conventions allows us to disregard some $\rho_{\mathtt{g}}$-graphs and thus to apply replacements (like for the evaluation rules in Figure 7) quite straightforwardly, since no variable capture needs to be considered.

In order to support the intuition, in the sequel we will sometimes provide a graphical representation of $\rho_{\mathtt{g}}$-graphs not including matching constraints. Roughly, any term without constraints can be represented as a tree in the obvious way, while a $\rho_{\mathtt{g}}$-graph $G\ [x_1 = G_1, \ldots, x_n = G_n]$ can be read as a letrec construct $\mathtt{letrec}\ x_1 = G_1, \ldots, x_n = G_n\ \mathtt{in}\ G$ and represented as a structure with sharing and cycles. Here the correspondence between a variable in the right-hand side of a rule and its binding occurrence in the pattern is represented by keeping the variable names (instead of using backpointers). The correspondence between a term and its graphical representation can be extend to general $\rho_{\mathtt{g}}$-graphs, possibly including matching constraints, as described in (Bertolissi, 2005). In this paper, we will use this correspondence only informally and for simple $\rho_{\mathtt{g}}$-graphs not containing match equations.

EXAMPLE 14 (Some $\rho_{\mathtt{g}}$-graphs). *For a graphical representation of the $\rho_{\mathtt{g}}$-graphs see Figure 6.*

1. *In the rule $(2 * x) \rightharpoonup ((y + y)\ [y = f(x)])$ the sharing in the right-hand side avoids the copying of the object instantiating $f(x)$, when the rule is applied to a $\rho_{\mathtt{g}}$-graph.*
2. *The $\rho_{\mathtt{g}}$-graph $cons(head(x), x)\ [x = cons(0, x)]$ represents an infinite list of zeros. Notice that the recursion variable x binds the occurrence of x in the right-hand side $cons(0, x)$ of the constraint and those in the term $cons(head(x), x)$ to which the constraint is applied.*
3. *The $\rho_{\mathtt{g}}$-graph $f(x, y)\ [x = g(y), y = g(x)]$ is an example of "twisted sharing" that can be expressed using mutually recursive constraints (to be read as a* `letrec` *construct). Here the preorder over variables is $x \geq y$ and $y \geq x$, and thus $x \equiv y$.*

For the purpose of this paper we restrict to *patterns* (used as left-hand sides of the abstractions and of the match equations) that are algebraic acyclic $\rho_{\mathtt{g}}$-graphs. For instance, the $\rho_{\mathtt{g}}$-graph $(f(y)\ [y = g(y)] \rightharpoonup a)$ is not allowed since the abstraction has a cyclic left-hand side.

BASIC RULES:

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

MATCHING RULES:

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

GRAPH RULES:

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

Figure 7. Small-step semantics of the ρ_{g}-calculus.

2.2. THE SMALL-STEP SEMANTICS OF ρ_{g}-CALCULUS

In the classical ρ-calculus, when reducing the application of a constraint to a term, *i.e.* a delayed matching constraint, the corresponding matching problem is solved and the resulting substitutions are applied at the meta-level of the calculus. In the $\rho_{\mathtt{x}}$-calculus, this reduction is decomposed into two phases, one computing substitutions and the other one describing the application of these substitutions. Matching computations leading from constraints to substitutions and the application of the substitutions are clearly separated and made explicit. In the ρ_{g}-calculus, the computation of the substitutions solving a matching constraint is performed explicitly 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 kept in the environment for a possible delayed application.

The complete set of evaluation rules of the ρ_{g}-calculus is presented in Fig. 7. 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. The rule (δ) distributes the application over the structures built with the "$\wr$" operator. The rule (ρ) triggers the application of a rewrite rule to a ρ_{g}-graph by applying the appropriate constraint to the right-hand side of the rule. For each of these rules, an additional rule dealing with the presence of constraints is considered. Without these rules the application of abstraction ρ_{g}-graphs like $R\ [x = R]\ f(a)$, where $R = f(y) \twoheadrightarrow x\ f(y)$ (that can encode a recursive application as in

Example 19) could not be reduced. An alternative solution would be to introduce appropriate distributivity rules but this approach is not considered in this paper.

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 syntactic matching, which is known to be decidable, but extensions to more elaborated theories are possible. Due to the assumptions on the left-hand sides of rewrite rules and constraints, we only need to decompose algebraic terms. The goal of this set of rules is to produce a constraint of the form $x_1 = G_1, \ldots, x_n = G_n$ starting from a match equation. Some replacements might be needed (as defined by the GRAPH RULES) as soon as the terms contain some sharing. The (*propagate*) rule flattens a list of constraints, thus propagating such constraints to a higher level. Note that, since left-hand sides of match equations are acyclic, there is no need for an evaluation rule propagating the constraints from the left-hand side of a match equation: the substitution and garbage collection rules can be used to obtain the same result. The algebraic terms are decomposed and the trivial constraints $K \ll K$ are eliminated. The rule (*solved*) transforms a matching constraint $x \ll G$ into a recursion equation $x = G$. The proviso asking that x is not defined elsewhere in the constraint is necessary in the case of matching problems involving non-linear constraints. For example, the constraint $x \ll a, x \ll b$ should not be reduced showing that the original (non-linear) matching problem has no solution.

The GRAPH RULES are inherited from the cyclic λ-calculus (Ariola and Klop, 1997). The (*sub*)stitution rules copy a ρ_{g}-graph associated to a recursion variable into a term inside the scope of the corresponding constraint. This is important to make a redex explicit (*e.g.* in $x\ a\ [x = a \twoheadrightarrow b]$) or to solve a match equation (*e.g.* in $a\ [a \ll x, x = a]$). As already mentioned, the rule (*acyclic sub*) allows one to make the copies only upwards *w.r.t.* the order defined on the variables of ρ_{g}-graphs. In the cyclic λ-calculus this is needed for the confluence of the system (see (Ariola and Klop, 1997) for a counterexample) and it will be also essential when proving the confluence of the ρ_{g}-calculus. Without this condition confluence is broken as one can see for the ρ_{g}-graph $z_1\ [z_1 = x \twoheadrightarrow z_2\ s(x), z_2 = y \twoheadrightarrow z_1\ s(y)]$ that reduces either to $z_1\ [z_1 = x \twoheadrightarrow z_1\ s(s(x))]$ or to $z_1\ [z_1 = x \twoheadrightarrow z_2\ s(x), z_2 = y \twoheadrightarrow z_2\ s(s(y))]$. The (*garbage*) rules get rid of recursion equations whose left-hand side variables do not appear in the scope of the equation itself (intuitively, they represent non-connected parts of the ρ_{g}-graph). Matching constraints are not eliminated, thus keeping trace of matching failures during an unsuccessful reduction. The (*black hole*) rules replace the undefined ρ_{g}-graphs, intuitively corresponding to self-loop graphs, with the constant $\bullet$.

We define the one step relations $\mapsto_{\mathcal{M}}$ and $\mapsto_{\mathcal{G}}$ and the many steps relations $\mapsto\!\!\!\!\!\rightarrow_{\mathcal{M}}$ and $\mapsto\!\!\!\!\!\rightarrow_{\mathcal{G}}$ *w.r.t.* the subset of MATCHING RULES and the whole set of rules of Figure 7 respectively. Note that all the evaluation steps are performed modulo the underlying theory associated to the conjunction operator "$\_, \_$".

With a view to a future efficient implementation of the calculus, it would be interesting to study suitable strategies that delay the application of the substitution rules (*external sub*) and (*acyclic sub*) to keep the sharing information as long as possible.

Basically, the idea consists of applying the substitution rules only if needed for generating new redexes for the BASIC RULES and to possibly unfreeze match equations where otherwise the computation of the matching is stuck. In addition, substitutions rules can be used to "remove" trivial recursion equations of the kind $x = y$.

DEFINITION 15. *The strategy* SharingStrat *allows to perform a step of reduction using the evaluation rules (external sub) or (acyclic sub) in a ρ_{g}-graph G only if:*

- *it instantiates an active variable by an abstraction or a structure, or*
- *it instantiates a variable in a stuck match equation,*
- *it instantiates a variable by a variable.*

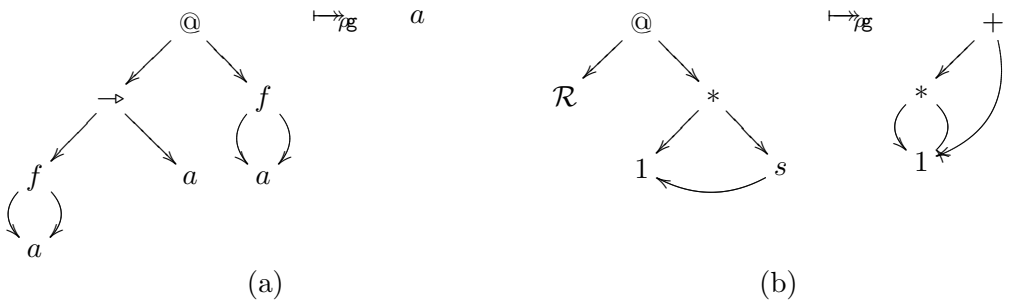


Figure 8. Examples of reductions.

This strategy is followed in the next examples. Note that the theory underlying the constraint conjunction operator "," is used during the reduction.

EXAMPLE 16 (A simple reduction with sharing). *A graphical representation is given in Figure 8(a).*

$$
\begin{array}{ll}
 & (f(x,x)\ [x=a] \rightarrowtriangle a)\ (f(y,y)\ [y=a]) \\
\mapsto_{p} & a\ [f(x,x)\ [x=a] \ll f(y,y)\ [y=a]] \\
\mapsto_{es} & a\ [f(a,a)\ [x=a] \ll f(y,y)\ [y=a]] \\
= & a\ [f(a,a)\ [x=a,\epsilon] \ll f(y,y)\ [y=a]] \quad (\textit{by neutral element axiom}) \\
\mapsto_{gc} & a\ [f(a,a)\ [\epsilon] \ll f(y,y)\ [y=a]] \\
\mapsto_{gc} & a\ [f(a,a) \ll f(y,y)\ [y=a]] \\
\mapsto_{p} & a\ [f(a,a) \ll f(y,y), y=a] \\
\mapsto_{dk} & a\ [a \ll y, a \ll y, y=a] \\
= & a\ [a \ll y, y=a] \qquad\qquad (\textit{by idempotency}) \\
\mapsto_{ac} & a\ [a \ll a, y=a] \\
\mapsto_{dk} & a\ [y=a] \\
= & a\ [y=a,\epsilon]\ \ (\textit{by neutral element axiom}) \\
\mapsto_{gc} & a\ [\epsilon] \\
\mapsto_{gc} & a
\end{array}
$$

EXAMPLE 17 (Multiplication). *Let us use an infix notation for the constant "$*$". The following ρ_g-term corresponds to the application of the rewrite rule $\mathcal{R} = x*s(y) \rightarrowtriangle (x*y+x)$ to the term $1*s(1)$ where the constant 1 is shared. The result is shown graphically in Figure 8(b).*

$$
\begin{array}{ll}
 & (x*s(y) \rightarrowtriangle (x*y+x))\ (z*s(z)\ [z=1]) \\
\mapsto_{p} & x*y+x\ [x*s(y) \ll (z*s(z)\ [z=1])] \\
\mapsto_{p} & x*y+x\ [x*s(y) \ll z*s(z), z=1] \\
\mapsto_{dk} & x*y+x\ [x \ll z, y \ll z, z=1] \\
\mapsto_{s} & x*y+x\ [x=z, y=z, z=1] \\
\mapsto_{es} & (z*z+z)\ [x=z, y=z, z=1] \\
\mapsto_{gc} & (z*z+z)\ [z=1]
\end{array}
$$

EXAMPLE 18 (Non-linearity). *The matching involving non-linear patterns can lead to a normal form that is either a constraint consisting only of recursion equations (which represents a successful matching) or a constraint that contains some match equations (representing a matching failure).*

$$
\begin{array}{ll}
 & f(y,y) \ll f(a,a) \\
\mapsto_{dk} & y \ll a\ \ (\textit{by idempotency}) \\
\mapsto_{s} & y=a
\end{array}
\qquad\qquad
\begin{array}{ll}
 & f(y,y) \ll f(a,b) \\
\mapsto_{dk} & y \ll a, y \ll b
\end{array}
$$

EXAMPLE 19 (Fixed point combinator). *Consider the term rewrite rule $R_Y = Y\ x \rightarrow x\ (Y\ x)$ which expresses the behaviour of the fixed point combinator Y of the λ-calculus. Given the a term*

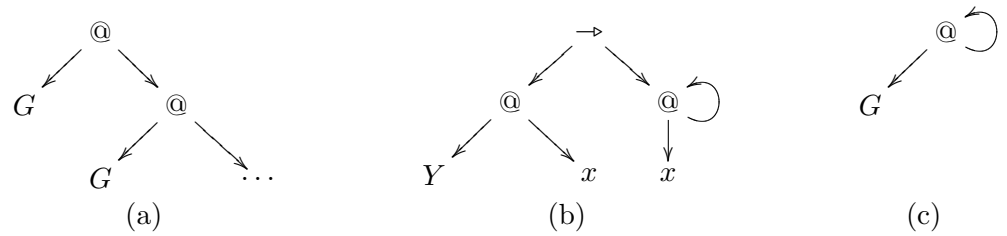


Figure 9. Example of reductions

t, we have the infinite rewrite sequence

$$Y\ t\ \rightarrow_{R_Y}\ t\ (Y\ t)\ \rightarrow_{R_Y}\ t\ (t\ (Y\ t))\ \rightarrow_{R_Y}\ \ldots$$

*which, in a sense which can be formalised (see (*Kennaway et al., 1991*;* Corradini, 1993*)), converges to the infinite term* $t\ (t\ (t\ (\ldots)))$.

We can represent the Y*-combinator in the* ρ_{g}*-calculus as the following term:*

$$Y \triangleq x_0\ [x_0 = x \twoheadrightarrow x\ (x_0\ x)].$$

If we define $R = x \twoheadrightarrow x\ (x_0\ x)$*, we have the following reduction:*

$$\begin{array}{ll} & Y\ G \\ \mapsto_{es} & (x \twoheadrightarrow x\ (x_0\ x))\ [x_0 = R]\ G \\ \mapsto_{p} & x\ (x_0\ x)\ [x \ll G, x_0 = R] \\ \mapsto_{s} & x\ (x_0\ x)\ [x = G, x_0 = R] \\ \mapsto_{es} & G\ (x_0\ G)\ [x = G, x_0 = R] \\ \mapsto_{gc} & G\ (x_0\ G)\ [x_0 = R] \\ \mapsto_{\not\mathsf{g}} & G(G\ldots(x_0\ G))\ [x_0 = R] \\ \mapsto_{\not\mathsf{g}} & \ldots \end{array}$$

Continuing the reduction, this will "converge" to the term of Figure 9(a).

*We can have a more efficient implementation of the same term reduction using a method introduced by Turner (*Turner, 1979*) that models the rule* R_Y *by means of the cyclic term depicted in Figure 9(b). This gives in the* ρ_{g}*-calculus the* ρ_{g}*-graph*

$$Y_T \triangleq x \twoheadrightarrow (z\ [z = x\ z])$$

The reduction in this case is the following:

$$\begin{array}{ll} & Y_T\ G \\ \mapsto_{p} & z\ [z = x\ z]\ [x \ll G] \\ \mapsto_{s} & z\ [z = x\ z]\ [x = G] \\ \mapsto_{es} & z\ [z = G\ z]\ [x = G] \\ \mapsto_{gc} & z\ [z = G\ z] \end{array}$$

The resulting ρ_{g}*-graph is depicted in Figure 9(c). If we "unravel", in the intuitive sense, this cyclic* ρ_{g}*-graph we obtain the infinite term shown in Figure 9(a).*

This intuitively means that a finite sequence of rewritings on cyclic ρ_{g}*-graphs can correspond to an infinite reduction sequence on the corresponding acyclic term.*

3. Confluence of the ρ_{g}-calculus

As mentioned before, ρ_{g}-graphs are grouped into equivalence classes defined according to the theory specified for the constraint conjunction operator and rewriting is performed over such classes. We have thus to analyse the properties of the corresponding relations modulo the

Strong normalization	SN	*no infinite* $\mapsto_S$ *reductions*
Diamond property modulo E	$D_\sim$	$\leftarrow\!\!\!\!\mapsto_S \cdot \mapsto_S \subseteq \mapsto_S \cdot \sim_E \cdot \leftarrow\!\!\!\!\mapsto_S$
Local confluence modulo E	$LCON_\sim$	$\leftarrow\!\!\!\!\mapsto_S \cdot \mapsto_S \subseteq \mapsto\!\!\!\!\!\!\rightarrow_S \cdot \sim_E \cdot \leftarrow\!\!\!\!\!\!\mapsto_S$
Confluence modulo E	$CON_\sim$	$\leftarrow\!\!\!\!\!\!\mapsto_S \cdot \mapsto\!\!\!\!\!\!\rightarrow_S \subseteq \mapsto\!\!\!\!\!\!\rightarrow_S \cdot \sim_E \cdot \leftarrow\!\!\!\!\!\!\mapsto_S$
Church-Rosser modulo E	$CR_\sim$	$\leftarrow\!\!\!\!\!\!\mapsto\!\!\!\!\!\!\rightarrow_{R\cup E} \subseteq \mapsto\!\!\!\!\!\!\rightarrow_S \cdot \sim_E \cdot \leftarrow\!\!\!\!\!\!\mapsto_S$
Commutation modulo E	$COM_\sim$	$\leftarrow\!\!\!\!\!\!\mapsto_{S_1} \cdot \mapsto\!\!\!\!\!\!\rightarrow_{S_2} \subseteq \mapsto\!\!\!\!\!\!\rightarrow_{S_2} \cdot \sim_E \cdot \leftarrow\!\!\!\!\!\!\mapsto_{S_1}$
Strong commutation modulo E	$SCOM_\sim$	$\leftarrow\!\!\!\!\mapsto_{S_1} \cdot \mapsto_{S_2} \subseteq \mapsto_{S_2}^{0/1} \cdot \sim_E \cdot \leftarrow\!\!\!\!\mapsto_{S_1}$
Compatibility with E	$CPB_\sim$	$\leftarrow\!\!\!\!\mapsto_S \cdot \sim_E \subseteq \sim_E \cdot \leftarrow\!\!\!\!\mapsto_S$
Coherence with E	$CH_\sim$	$\leftarrow\!\!\!\!\!\!\mapsto_S \cdot \sim_E \subseteq \mapsto\!\!\!\!\!\!\rightarrow_S \cdot \sim_E \cdot \leftarrow\!\!\!\!\!\!\mapsto_S$

Figure 10. Properties of rewriting modulo E.

underlying theory of the constraint conjunction operator. In the next section we give the formal definition of rewriting modulo and the definition of several classical properties of corresponding term rewrite systems. We also introduce some propositions that will be used in the subsequent sections in order to prove the confluence of ρ_{g}-calculus.

3.1. Equational rewriting

Given a set of equations E over a set of terms $\mathcal{T}$, we denote by $\sim_E^1$ the one step equality, *i.e.* for any context $\mathsf{Ctx}\{\Box\}$ and any substitution σ, if $T_1 = T_2$ is an equation in E then $\mathsf{Ctx}\{\sigma(T_1)\} \sim_E^1 \mathsf{Ctx}\{\sigma(T_2)\}$. Let $\sim_E$ be the symmetric and transitive closure of $\sim_E^1$ over the set $\mathcal{T}$; two terms T_1 and T_2 are said equivalent modulo E if $T_1 \sim_E T_2$.

We define next a notion of rewriting where the rewrite rules act over equivalence classes of terms modulo $\sim_E$. This approach is rather general but may be very inefficient since, in order to reduce a given term, all the terms in the same equivalence class must be taken into consideration and such a class can be quite large or even infinite. A possible refinement of this reduction relation is studied in (Peterson and Stickel, 1981; Jouannaud and Kirchner, 1984) and called rewriting modulo E. Using this notion of reduction, the rules apply to terms rather than to equivalence classes, but matching modulo E is performed at each step of the reduction.

DEFINITION 20 (E-class rewriting and rewriting modulo E). *Let $\mathcal{R}$ be a set of rewrite rules and E be a set of equations over a set of terms $\mathcal{T}$. Let $L \to R \in \mathcal{R}$ be a rewrite rule and σ a substitution. Then*

1. *a term T_1 E-class rewrites to a term T_2, denoted $T_1 \mapsto_{\mathcal{R}/E} T_2$ iff $T_1 \sim_E \mathsf{Ctx}\{\sigma(L)\}$ and $T_2 \sim_E \mathsf{Ctx}\{\sigma(R)\}$.*
2. *a term T_1 rewrites modulo E to a term T_2, denoted $T_1 \mapsto_{\mathcal{R},E} T_2$ iff $T_1 = \mathsf{Ctx}\{T\}$ with $T \sim_E \sigma(L)$ and $T_2 = \mathsf{Ctx}\{\sigma(R)\}$.*

Given a rewriting relation $\mapsto_S$ we denote by $\mapsto\!\!\!\!\!\!\rightarrow_S$ its reflexive and transitive closure, by $\leftarrow\!\!\!\!\!\!\mapsto\!\!\!\!\!\!\rightarrow_S$ its symmetric, reflexive and transitive closure. A zero or one step reduction is denoted by $\mapsto_S^{0/1}$. In Figure 10 we give the definitions of several classical properties of term rewrite systems, some generalised to rewriting modulo a set of axioms. We will write $PROP(S)$ if a property $PROP$ holds for a relation $\mapsto_S$. In Figure 11 some of these properties as represented graphically.

It is easy to see that $CPB_\sim$ with $\sim_E^1$ is equivalent to $CPB_\sim$ with $\sim_E$ and that $CPB_\sim$ implies $CH_\sim$. Several other relationships between the above properties are stated and proved in (Ohlebusch, 1998) but we recall here only two propositions about confluence. Notice that, compared to standard rewriting, in order to have confluence for rewriting modulo a set of equations E, an additional compatibility property of the rewrite system *w.r.t.* the congruence relation generated by E comes into play.

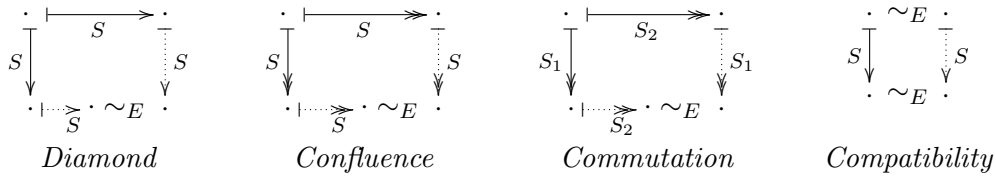


Figure 11. Properties of rewriting modulo $\sim_E$.

PROPOSITION 21. *(*Ohlebusch, 1998*)*

1. *A terminating relation S locally confluent modulo E and compatible with E is confluent modulo E.*
2. *The union of two relations S_1 and S_2 commuting modulo E, both confluent modulo E and compatible with E, is confluent modulo E.*

In the following we will need also two properties that ensure the commutation of two sets of rewrite rules:

PROPOSITION 22.

1. *If S_1 and S_2 verify the property $\leftarrowtail_{S_1} \cdot \twoheadrightarrow_{S_2} \subseteq \twoheadrightarrow_{S_2} \cdot \sim_E \cdot \twoheadleftarrow_{S_1}$ (denoted $PR_{\sim}(S_1, S_2)$) and are compatible with E, then S_1 and S_2 commute modulo E.*
2. *Two strongly commuting relation S_1 and S_2 compatible with E commute modulo E.*

Proof: Point (1) is proved by induction on the number of steps of S_1. Point (2) follows by using (1) and an induction on the number of steps of S_2. □

3.2. General presentation

The confluence for higher-order systems dealing with non-linear matching is a difficult issue since we usually obtain non-joinable critical pairs as shown by Klop in the setting of the λ-calculus (Klop, 1980). Klop's counterexample can be encoded in the ρ-calculus (Wack, 2003) showing that the non-linear ρ-calculus is not confluent, if no evaluation strategy is imposed on the reductions. The counterexample is still valid when generalising the ρ-calculus to the ρ_{g}-calculus, therefore in the following we consider a version of the ρ_{g}-calculus with some linearity assumptions.

DEFINITION 23 (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 $FV(\mathcal{L}_i) \cap FV(\mathcal{L}_j) = \emptyset$ 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 $FV(L_i) \cap FV(L_j) = \emptyset$ for $i \neq j$. The* linear ρ_{g}-calculus *is the ρ_{g}-calculus where all the patterns in the left-hand side of abstractions and all constraints are linear.*

In the general ρ_{g}-calculus, the operator "$\_,\_$" is supposed to be associative, commutative and idempotent, with the empty list of constraints ϵ as neutral element. However, in the linear ρ_{g}-calculus, idempotency is not needed since constraints of the form $x \ll G, x \ll G$ are not allowed (and cannot arise from reductions). Therefore, in the linear 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.

Following the notation in Definition 20, the relation induced over $AC1$-equivalence classes is written $\mapsto_{\rho_g/AC1}$. Concretely, in most of the proofs we will use the notion of rewriting modulo $AC1$ (Peterson and Stickel, 1981), denoted $\mapsto_{\rho_g,AC1}$. On the one hand, this notion of rewriting is more convenient, from a computational point of view, than $AC1$-class rewriting. On the other hand, as we will see in Section 3.3, under suitable assumptions satisfied by our calculus, the confluence of the relation $\rho_g, AC1$ implies the confluence of the $\rho_g/AC1$ relation.

According to the definition of $\mapsto_{\rho_g,AC1}$, matching modulo $AC1$ is potentially performed at each evaluation step. We mention that matching modulo $AC1$ may lead to infinitely many solutions, but the complete set of solution is finitary and has as canonical representative the solution in which terms are normalised *w.r.t.* the neutral element (Kirchner, 1990).

The confluence proof is quite elaborated and we decompose it in a number of lemmata to achieve the final result. Its complexity is mainly due to the non-termination of the system and to the fact that equivalence modulo $AC1$ on ρ_{g}-graphs has to be considered throughout the proof.

We start by proving a fundamental compatibility lemma showing that the ρ_{g}-calculus rewrite relation is particularly well-behaved *w.r.t.* the congruence relation $AC1$. This lemma ensures that if there exists a rewrite step from a ρ_{g}-graph G, then the "same" step can be performed starting from any term $AC1$-equivalent to G.

LEMMA 24 (Compatibility of ρ_g). *Compatibility with AC1 holds for any rule r of the ρ_{g}-calculus.*

$$\hookleftarrow_{r,AC1} \cdot \sim_{AC1} \subseteq \sim_{AC1} \cdot \hookleftarrow_{r,AC1}$$

Proof: By case analysis on the rules of the ρ_{g}-calculus. Consider, for instance, the diagram for the (*acyclic sub*) rule with a commutation step.

$$\begin{array}{ccc} G\ [G_0 \lll \mathsf{Ctx}\{y\}, y = G_1, F] & \sim^1_{AC1} & G\ [y = G_1, G_0 \lll \mathsf{Ctx}\{y\}, F] \\ \Big\downarrow{\scriptstyle ac,AC1} & & \Big\downarrow{\scriptstyle ac,AC1} \\ G\ [G_0 \lll \mathsf{Ctx}\{G_1\}, y = G_1, F] & \sim_{AC1} & G\ [y = G_1, G_0 \lll \mathsf{Ctx}\{G_1\}, F] \end{array}$$

A different order of the constraints in the list do not prevent the application of the (*acyclic sub*) rule, thanks to the fact that matching is performed modulo $AC1$. Moreover, the extension variable E in the definition of the (*acyclic sub*) rule ensures the applicability of the rule to ρ_{g}-graphs having an arbitrary number of constraints in the list. In particular, the extension variable E can be instantiated by ϵ if the term to reduce is simply $G\ [G_0 \lll \mathsf{Ctx}\{y\}, y = G_1]$. In this case the application of the rule is possible since there exists a match of the (*acyclic sub*) rule in the term $[G_0 \lll \mathsf{Ctx}\{y\}, y = G_1, \epsilon]$, equivalent to the given term using the neutral element axiom.

The diagram can thus be easily closed. The same reasoning can be applied for the other rules of the ρ_{g}-calculus. The extension to several steps of $\sim_{AC1}$ trivially holds. □

We point out that since compatibility holds for any evaluation rule of the ρ_{g}-calculus, then it also holds for any subset of rules, including the entire set of rules of ρ_{g}-calculus.

For proving the confluence of $\mapsto\!\!\!\!\!\rightarrow_{\rho_g,AC1}$ we use a technique inspired by the one adopted for the confluence of the cyclic λ-calculus (Ariola and Klop, 1997). The larger number of evaluation rules of the ρ_{g}-calculus and the explicit treatment of the congruence relation on ρ_{g}-graphs make the proof for the ρ_{g}-calculus much more elaborated.

The main idea of the proof is to split the rules into two subsets, to show separately their confluence and then to prove the confluence of the union using a commutation lemma for the two sets of rules. The ρ_{g}-calculus rules are thus split into the following two subsets:

– Σ-*rules*, including the substitution rules (*external sub*) and (*acyclic sub*), plus the (δ) rule;

– τ-*rules*, including all the remaining rules.

The Σ-rules include the substitution rules which represent the "non-terminating part" of the ρ_{g}-calculus. The (δ) rule is also included in the Σ-rules, although it could be safely added to the τ-rules keeping this set of rules terminating. This choice motivated by the fact that, because of its non-linearity, adding the (δ) rule to the τ-rules would have caused relevant problems in the proof of the final commutation lemma (Lemma 42).

The confluence proof of $\mapsto\!\!\!\!\!\rightarrow_{\rho_g,AC1}$ is structured in three parts.

– *confluence modulo AC1 of the relation induced by the τ-rules*
This is done by using the fact that a relation strongly normalising and locally confluent modulo *AC1* is confluent modulo *AC1* if the compatibility property holds (see Proposition 21). To prove the strong normalisation a polynomial interpretation on the ρ_{g}-calculus is used. Local confluence modulo *AC1* is easy to prove by analysis of the critical pairs.

– *confluence modulo AC1 of the relation induced by the Σ-rules*
This is the more complex part of the proof. The idea is to exploit the *complete development method* of the λ-calculus, defining a terminating version of the relation induced by the Σ rules (the development) and using its properties for deducing the confluence of the original rewrite relation.

– *confluence modulo AC1 of the relation induced by the union of the two sets*
General confluence holds since we can prove the commutation modulo *AC1* of the two relations.

From the confluence of the relation $\mapsto\!\!\!\!\!\rightarrow_{\rho_g,AC1}$ we can deduce the confluence of the relation $\mapsto\!\!\!\!\!\rightarrow_{\rho_g/AC1}$ acting on *AC1*-equivalence classes of ρ_{g}-graphs. This is a consequence of the fact that the compatibility with *AC1* property holds for the rules of the ρ_{g}-calculus.

In the following, to simplify the notation we will simply write *AC1* or $\sim$ for $\sim_{AC1}$ and $\mapsto_R$ for $\mapsto_{R,AC1}$, where R may be any subset of rules of the ρ_{g}-calculus.

The outline of the proof is depicted in Figure 12, where all the lemmata are mentioned, except for the the compatibility lemma which is left implicit, since it is used for almost all the intermediate results.

3.3. The complete confluence proof

Confluence modulo AC_ϵ for the τ-rules

The confluence modulo *AC1* for the relation $\mapsto\!\!\!\!\!\rightarrow_\tau$ induced by the τ-rules is proved by showing that this relation is strongly normalising and locally confluent modulo *AC1*. In the first part of the section we prove strong normalisation for $\mapsto\!\!\!\!\!\rightarrow_\tau$ by using a reduction order based on a polynomial interpretation on the ρ_{g}-graphs. In the second part of the section, the local confluence modulo *AC1* is proved for the relation $\mapsto\!\!\!\!\!\rightarrow_\tau$ by case analysis of the critical pairs. On the basis of these results, we can then conclude the confluence of the relation $\mapsto\!\!\!\!\!\rightarrow_\tau$.

We start by showing that $\mapsto\!\!\!\!\!\rightarrow_\tau$ is strongly normalising. In order to do that, we define a polynomial interpretation on the the ρ_{g}-calculus syntax.

DEFINITION 25 (Polynomial interpretation). *We consider the following polynomial interpretation of ρ_{g}-graphs (assuming the standard order on natural numbers):*

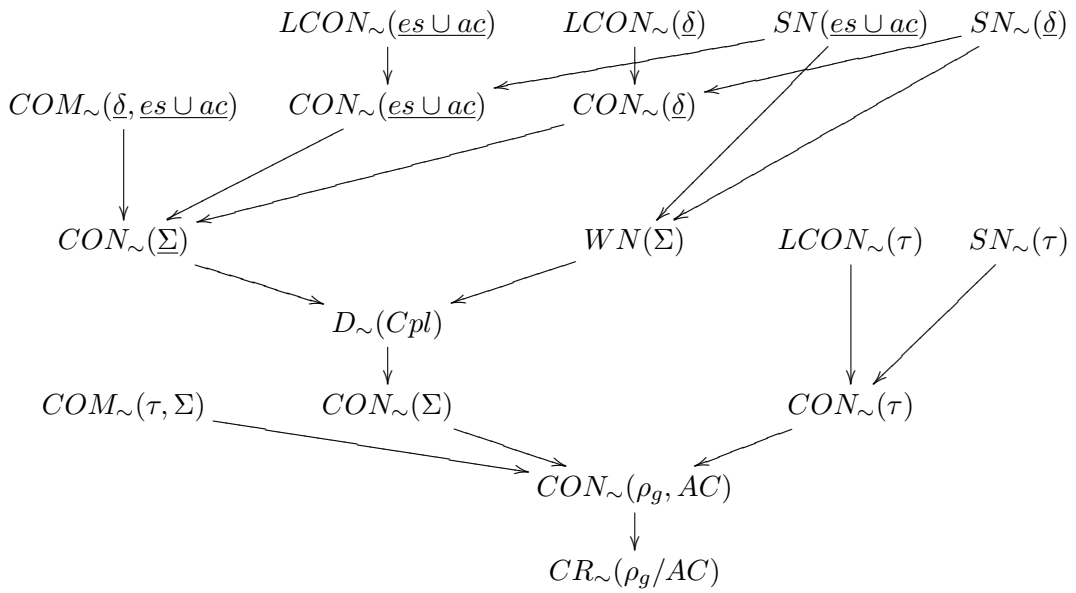


Figure 12. Confluence proof scheme

$$
\begin{array}{lcl}
Size(\epsilon) & = & 0 \\
Size(\bullet) & = & 1 \\
Size(x) & = & Size(f) \;=\; 2 \textit{ for all } x \in \mathcal{X} \textit{ and } f \in \mathcal{K} \setminus \{\bullet\} \\
Size(G_1 * G_2) & = & Size(G_1) + Size(G_2) + 1 \textit{ where } * \in \{\ \_ \wr \_,\ \_ \_,\ \_ \ll \_\} \\
Size(G_1 \twoheadrightarrow G_2) & = & Size(G_1) + Size(G_2) + 2 \\
Size(G\ [E]) & = & Size(G) + Size(E) + 1 \\
Size(E, E') & = & Size(E) + Size(E') \\
Size(x = G) & = & Size(x) + Size(G)
\end{array}
$$

We point out that the polynomial interpretation is compatible *w.r.t.* to neutrality of ϵ for the constraint conjunction operator. In fact $E, \epsilon = E$ and $Size(E, \epsilon) = Size(E) + Size(\epsilon) = Size(E) + 0 = Size(E)$. Similarly, it is compatible *w.r.t.* the associativity and commutativity of the conjunction and *w.r.t.* α-conversion. Moreover, function $Size(\cdot)$ can be easily seen to be monotonic and closed under contexts.

LEMMA 26 (Context closure). *Let G_1 and G_2 be two ρ_{g}-graphs. If $Size(G_1) > Size(G_2)$ then $Size(\mathsf{Ctx}\{G_1\}) > Size(\mathsf{Ctx}\{G_2\})$, for all contexts $\mathsf{Ctx}\{\Box\}$.*

Proof: Since the addition is increasing on naturals, the lemma is clearly satisfied. □

We next show that for all the rules in τ, the polynomial interpretation of the left-hand side is bigger than that of the right-hand side for any substitution of the rule (meta-)variables by positive naturals. As a consequence, we get the termination of the $\mapsto\!\!\!\!\rightarrow_\tau$ relation.

LEMMA 27 (SN(τ)). *The relation $\mapsto\!\!\!\!\rightarrow_\tau$ is strongly normalising.*

Proof: Clearly $Size(\cdot)$ associates a natural number to any constraint and ρ_{g}-graph (precisely $Size(\mathcal{C}) \geq 0$ for any constraint $\mathcal{C}$ and $Size(G) \geq 1$ for any ρ_{g}-graph G). Now, it is not difficult to check that for any rule $L \to R$ in τ we have $Size(L) > Size(R)$ for all interpretations of the meta-variables of L and R over natural numbers. Hence, by Lemma 26, for all ρ_{g}-graphs G_1 and

G_2 such that $G_1 \mapsto_\tau G_2$ we have $Size(G_1) > Size(G_2)$. Therefore the relation $\mapsto\!\!\!\!\!\rightarrow_\tau$ is strongly normalising.

□

We next prove the local confluence modulo *AC1* of the relation $\mapsto\!\!\!\!\!\rightarrow_\tau$. This is done by inspection of the critical pairs generated by the τ-rules.

LEMMA 28 (LCON$_\sim(\tau)$). *The relation $\mapsto\!\!\!\!\!\rightarrow_\tau$ is locally confluent modulo AC1.*

Proof: The proof is done by inspecting the possible critical pairs. The (*decompose*) rule and the (*garbage*) rule generate only trivial critical pairs with the other τ-rules. The (ρ) rule generates a joinable critical pair with the (*black hole*) rule as shown in the next diagram:

$$\begin{array}{ccc} (P \twoheadrightarrow \mathsf{Ctx}\{x\})\ [x =_\circ x, E]\ G_3 & \xrightarrow[bh]{} & (P \twoheadrightarrow \mathsf{Ctx}\{\bullet\})\ [x =_\circ x, E]\ G_3 \\ \downarrow \rho & & \downarrow \rho \\ \mathsf{Ctx}\{x\}\ [P \ll G_3, x =_\circ x, E] & \xrightarrow[bh]{} & \mathsf{Ctx}\{\bullet\}\ [P \ll G_3, x =_\circ x, E] \end{array}$$

The (*propagate*) rule generates a joinable critical pair with the (*black hole*) rule:

$$\begin{array}{ccc} P \ll \mathsf{Ctx}\{x\}\ [x =_\circ x, E] & \xrightarrow[bh]{} & P \ll \mathsf{Ctx}\{\bullet\}\ [x =_\circ x, E] \\ \downarrow p & & \downarrow p \\ P \ll \mathsf{Ctx}\{x\}, x =_\circ x, E & \xrightarrow[bh]{} & P \ll \mathsf{Ctx}\{\bullet\}, x =_\circ x, E \end{array}$$

Finally, the (*solved*) rule generates a joinable critical pair with the (*black hole*) rule:

$$\begin{array}{ccc} y \ll \mathsf{Ctx}\{x\}, x =_\circ x, E & \xrightarrow[bh]{} & y \ll \mathsf{Ctx}\{\bullet\}, x =_\circ x, E \\ \downarrow s & & \downarrow s \\ y = \mathsf{Ctx}\{x\}, x =_\circ x, E & \xrightarrow[bh]{} & y = \mathsf{Ctx}\{\bullet\}, x =_\circ x, E \end{array}$$

□

As in standard term rewriting, we can use local confluence and strong normalisation to conclude about the confluence of a relation. By the fact that we consider (local) confluence modulo a set of equations, an additional compatibility property is needed to conclude the desired result.

PROPOSITION 29 (CON$_\sim(\tau)$). *The relation $\mapsto\!\!\!\!\!\rightarrow_\tau$ is confluent modulo AC1.*

Proof: By Proposition 21 using Lemma 24, Lemma 27 and Lemma 28. □

Confluence modulo *AC1* for the Σ-rules

We present in this section the more elaborated part of the proof, namely the proof of confluence modulo *AC1* for the relation $\mapsto\!\!\!\!\!\rightarrow_\Sigma$ induced by the Σ-rules. The difficulties in this issue arise from the fact that the rewrite relation $\mapsto\!\!\!\!\!\rightarrow_\Sigma$ is not strongly normalising. In particular, notice that the rewrite relations induced by the substitution rules are both not terminating in the presence of cycles:

$x\ [x = f(y), y = g(y)] \mapsto_{ac} x\ [x = f(g(y)), y = g(y)] \mapsto_{ac} \ldots$
$y\ [y = g(y)] \mapsto_{es} g(y)\ [y = g(y)] \mapsto_{es} \ldots$

Consequently, the techniques used in the previous section for the $\mapsto\!\!\!\!\rightarrow_{\mathcal{T}}$ relation do not apply to prove confluence in this case. Taking inspiration from the confluence proof of the cyclic λ-calculus in (Ariola and Klop, 1997), we use the so-called *complete development method* of the λ-calculus, adapting it to the relation $\mapsto\!\!\!\!\rightarrow_{\Sigma}$. The idea of this proof technique consists, first, in defining a new rewrite relation *Cpl* with the same transitive closure as the $\mapsto\!\!\!\!\rightarrow_{\Sigma}$ relation and, secondly, in proving the diamond property modulo *AC1* of this relation. We can the conclude on the confluence of the original $\mapsto\!\!\!\!\rightarrow_{\Sigma}$ relation.

Intuitively, a step of *Cpl* rewriting on a term G consists of the complete reduction of a set of redexes initially fixed in G. In other words, some redexes are marked in G and a complete development of these redexes is performed by the *Cpl* relation. Concretely, an underlining operator is used to mark some redexes and then the reductions on underlined redexes are performed using the following underlined versions of the Σ-rules:

$$
\begin{array}{lll}
(\underline{\textit{external sub}}) & \mathsf{Ctx}\{\underline{y}\}\ [y = G, E] & \rightarrow_{\underline{es}}\ \mathsf{Ctx}\{G\}\ [y = G, E] \\
(\underline{\textit{acyclic sub}}) & G\ [G_0 \lll \mathsf{Ctx}\{\underline{y}\}, y = G_1, E] & \rightarrow_{\underline{ac}}\ G\ [G_0 \lll \mathsf{Ctx}\{G_1\}, y = G_1, E] \\
 & & \textit{if } x > \underline{y},\ \forall x \in \mathcal{FV}(G_0) \\
(\underline{\delta}) & (G_1 \underline{\wr} G_2)\ G_3 & \rightarrow_{\underline{\delta}}\ G_1\ G_3 \wr G_2\ G_3 \\
 & (G_1 \underline{\wr} G_2)\ [E]\ G_3 & \rightarrow_{\underline{\delta}}\ (G_1\ G_3 \wr G_2\ G_3)\ [E]
\end{array}
$$

We call the new rewrite relation $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ and the associated calculus $\underline{\Sigma}$-calculus. Terms belonging to the $\underline{\Sigma}$-calculus are ρ_{g}-graphs in which some recursion variables belonging to a $\underline{\Sigma}$-redex are underlined.

EXAMPLE 30 (Terms of the $\underline{\Sigma}$-calculus).

- $\underline{x}\ [x = f(x)]$ *is a legal term.*
- $x\ [x = f(\underline{x})]$ *is not a legal term, since* $x \equiv \underline{x}$ *and thus the proviso for the application of the* ($\underline{\textit{acyclic sub}}$) *rule is not verified.*
- *Similarly,* $f(x)\ [x \ll f(\underline{y}), y = g(\underline{z}, \underline{z}), z = \underline{y}]$ *is not a legal term, since* $x > \underline{y}$ *but* $\underline{z} \equiv \underline{y}$.
- $f(x)\ [x \ll f(\underline{y}), y = g(\underline{z}, \underline{z}), z = y]$ *is a legal term, since here* $x > \underline{y}$ *and also* $y > \underline{z}$.

The *Cpl* rewrite relation is then defined as follows.

DEFINITION 31 (*Cpl* relation). *Given the* ρ_{g}*-graphs* G_1 *and* G_2 *in the* Σ*-calculus, we have that* $G_1 \mapsto\!\!\!\!\rightarrow_{Cpl} G_2$ *if there exists an underlining* G_1' *of* G_1 *such that* $G_1' \mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}} G_2$ *and* G_2 *is in normal form w.r.t. the relation* $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$.

EXAMPLE 32 (Reductions in the $\underline{\Sigma}$-calculus).

- *The term* $x\ [x = f(\underline{y}), y = g(y)]$ *reaches the* $\underline{\Sigma}$ *normal form* $x\ [x = f(g(y)), y = g(y)]$ *in one* ($\underline{ac}$)*-step.*
- *We have the reduction* $G_1 = x\ [f(x, y) \ll f(\underline{z}, \underline{z}), z = g(\underline{w}), w = a] \mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}} x\ [f(x, y) \ll f(\underline{z}, \underline{z}), z = g(a), w = a] \mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}} x\ [f(x, y) \ll f(g(a), g(a)), z = g(a), w = a] = G_2$ *and thus* $G_1 \mapsto\!\!\!\!\rightarrow_{Cpl} G_2$.

To ensure the fact that for every possible underlining of redexes in a ρ_{g}-graph G_1 we have a corresponding *Cpl* reduction, we need to prove that for every underlined term there exists a normal form *w.r.t.* the $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ reduction, *i.e.* we must prove that $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ is weakly normalising. To this aim, we prove first that the relations induced by the (δ) rule and the underlined substitution rules separately are strongly normalising.

LEMMA 33. *$SN(\underline{\delta})$ and $SN(\{\underline{es}, \underline{ac}\})$ hold.*

Proof: The strong normalisation of the relation induced by the $(\underline{\delta})$ rule can be proved by using the multiset path ordering induced by the following precedence on the operators of the ρ_{g}-calculus:

$$\_\,\_ \succ \_\wr\_ \succ \_[\_] \succ \_\ll\_ \succ \_,\_ \succ \_=\_$$

For proving the termination of the relation induced by $\{\underline{es}, \underline{ac}\}$ we exploit a technique inspired by (Ariola and Klop, 1997). We define the weight associated to a term of the $\underline{\Sigma}$-calculus as the multiset of all its underlined recursion variables, ordered by standard multiset ordering induced by the ordering $>$ among recursion variables (see Definition 10). Then we show that the weight decreases during the reduction. We analyse the two different cases:

- If we substitute an underlined recursion variable by a term containing no underlined variables, then the weight trivially decreases. For example $\underline{x}\ [x = f(x)]$ has weight $\{\underline{x}\}$ while its reduct $f(x)\ [x = f(x)]$ has weight $\emptyset$.
- If we substitute an underlined recursion variable $\underline{x}$ by a term containing one or more recursion variables $\underline{y_1}, \ldots, \underline{y_n}$, then we have $\underline{x} > \underline{y_i}$ for all $i = 1, \ldots, n$ otherwise the term would not be a legal $\underline{\Sigma}$-calculus term. It follows that the multiset of the reduced term is smaller than the one associated to the initial term. Consider for example the ρ_{g}-graph $G = x\ [x = \mathsf{C}_0\{\underline{y_1}\}, y_1 = \mathsf{C}_1\{\underline{y_2}\}, y_2 = G']$ and the reduction

$$G \mapsto_{\underline{ac}} x\ [x = \mathsf{Ctx}_0\{\mathsf{C}_1\{\underline{y_2}\}\}, y_1 = \mathsf{C}_1\{\underline{y_2}\}, y_2 = G']$$

  The multiset associated to G is $\{\underline{y_1}, \underline{y_2}\}$. By the definition of the order on recursion variables, we have $x > y_1$ and $y_1 > y_2$, so the multiset $\{\underline{y_2}, \underline{y_2}\}$ associated to the ρ_{g}-graph obtained after the reduction is smaller. Notice that $y_1 \neq y_2$ otherwise the proviso of the (*acyclic sub*) rule would not be satisfied and G would not be a legal term. For the same reason, no $\underline{y_1}$ is allowed on the right-hand side of the recursion equation for y_2.

□

PROPOSITION 34 (WN($\underline{\Sigma}$)). *The relation $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ is weakly normalising.*

Proof: Given any $\underline{\Sigma}$-term, a normal form can be reached by using the rewriting strategy where $(\underline{\delta})$ has greater priority than $(\{\underline{es}, \underline{ac}\})$. By Lemma 33 we know that the relations induced by $(\underline{\delta})$ and $(\{\underline{es}, \underline{ac}\})$ are strongly normalising. Observe that the $(\{\underline{es}, \underline{ac}\})$ induced relation does not generate $(\underline{\delta})$ redexes. Hence we can normalise a term G first *w.r.t.* the $(\underline{\delta})$ induced relation and then *w.r.t.* the $(\{\underline{es}, \underline{ac}\})$ induced relation obtaining thus a finite reduction of G. □

The next goal is to prove the diamond property of the *Cpl* relation. In order to do this, the confluence modulo *AC1* of the $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ relation is needed. Since we know that the relations induced by $(\underline{\delta})$ and $(\{\underline{es}, \underline{ac}\})$ are both strongly normalising, we prove their local confluence modulo *AC1* by analysis of the critical pairs and then we conclude on their confluence modulo *AC1*. The confluence modulo *AC1* of the $\mapsto\!\!\!\!\rightarrow_{\underline{\Sigma}}$ relation will then follow using a commutation lemma.

LEMMA 35. *$LCON_{\sim}(\underline{\delta})$ and $LCON_{\sim}(\{\underline{es}, \underline{ac}\})$ hold.*

Proof: We proceed by analysis of the critical pairs. The critical pairs of the $(\underline{\delta})$ rule with itself are trivial. Among the critical pairs of the (*external sub*) and (*acyclic sub*) rules, we show next the diagrams for two interesting cases. We consider the case where $\lll$ is equal to $=$. The case where $\lll$ is $\ll$ can be treated in the same way. To ease the notation, from now on we will write just $\mathsf{C}_i\{G\}$ for a context $\mathsf{Ctx}_i\{G\}$ in the critical pairs diagrams.

– Consider the critical pair generated from a term having a list of constraints containing two non-disjoint ($\underline{ac}$) redexes. Notice that the recursion variable $\underline{z}$ can be duplicated by the first ($\underline{ac}$)-step.

$$\begin{array}{ccc}
G_0\ [y = \mathsf{C}_1\{\underline{x}\}, x = \mathsf{C}_2\{\underline{z}\}, z = G_1] & \xrightarrow[\underline{ac}]{} & G_0\ [y = \mathsf{C}_1\{\underline{x}\}, x = \mathsf{C}_2\{G_1\}, z = G_1] \\
\downarrow \underline{ac} & & \vdots\ \underline{ac} \\
G_0\ [y = \mathsf{C}_1\{\mathsf{C}_2\{\underline{z}\}\}, x = \mathsf{C}_2\{\underline{z}\}, z = G_1] & \twoheadrightarrow_{\underline{ac}} & G_0\ [y = \mathsf{C}_1\{\mathsf{C}_2\{G_1\}\}, x = \mathsf{C}_2\{G_1\}, z = G_1]
\end{array}$$

– Consider the critical pair in which the term duplicated by an ($\underline{es}$) step contains an ($\underline{ac}$) redex. Notice that we need both the substitution rules, *i.e.* $\underline{ac} \cup \underline{es}$, to close the diagram.

$$\begin{array}{ccc}
\mathsf{C}_0\{\underline{y}\}\ [y = \mathsf{C}_1\{\underline{x}\}, x = \mathsf{C}_2\{x\}] & \xrightarrow[\underline{es}]{} & \mathsf{C}_0\{\mathsf{C}_1\{\underline{x}\}\}\ [y = \mathsf{C}_1\{\underline{x}\}, x = \mathsf{C}_2\{x\}] \\
\downarrow \underline{ac} & & \vdots\ \underline{ac \cup es} \\
\mathsf{C}_0\{\underline{y}\}\ [y = \mathsf{C}_1\{\mathsf{C}_2\{x\}\}, x = \mathsf{C}_2\{x\}] & \twoheadrightarrow_{\underline{ac}} & \mathsf{C}_0\{\mathsf{C}_1\{x\}\}\ [y = \mathsf{C}_1\{\mathsf{C}_2\{x\}\}, x = \mathsf{C}_2\{x\}]
\end{array}$$

□

At this point, it is worth noticing that the local compatibility with *AC1* holds for the underlined version of the rules. This property, together with the local confluence modulo *AC1* and the strong normalisation for the relations induced by the rules ($\underline{\delta}$) and ($\{\underline{es}, \underline{ac}\}$) is sufficient to prove their confluence.

LEMMA 36. *$CPB_\sim(\underline{\delta})$ and $CPB_\sim(\{\underline{es}, \underline{ac}\})$ hold.*

Proof: By Lemma 24 we know that this property holds for the original version of the rules, without underlining. Since equivalence steps in the *AC1* theory have no effect with respect to the underlining, we can conclude that the lemma is true also for the underlined rules. □

LEMMA 37. *$CON_\sim(\underline{\delta})$ and $CON_\sim(\{\underline{es}, \underline{ac}\})$ hold.*

Proof: By Proposition 21 using Lemma 33, Lemma 35 and Lemma 36. □

After having proved the confluence modulo *AC1* of the relations induced by the two subsets of rules independently, following Proposition 21, we need a commutation lemma to be able to conclude about the confluence of the relation induced by the union of the two subsets.

LEMMA 38. *$COM_\sim(\underline{\delta}, \{\underline{es}, \underline{ac}\})$ holds.*

Proof: General commutation is not easy to prove, thus we prove a simpler property which implies commutation. By Lemma 36, we know that the compatibility property holds for our relations. both of them can duplicate redexes of the other one. Nevertheless, the relations do not interfere with each other, in the sense that, for example, a ($\underline{\delta}$) redex will still be present (possibly duplicated) after one or several steps of ($\{\underline{es}, \underline{ac}\}$). Therefore, we will use Proposition 22 and we need simply to verify the property

$$\leftarrow\!\dashv_{\{\underline{es},\underline{ac}\}} \cdot \mapsto\!\!\!\twoheadrightarrow_{\underline{\delta}} \ \subseteq\ \mapsto\!\!\!\twoheadrightarrow_{\underline{\delta}} \cdot \sim_E \cdot \twoheadleftarrow\!\dashv_{\{\underline{es},\underline{ac}\}}$$

We proceed by analysis of the critical pairs. We discuss explicitly only the critical pairs between the ($\underline{\delta}$) rule and the ($\underline{es}$) rule, since the treatment for the critical pairs between the ($\underline{\delta}$) rule and the ($\underline{ac}$) rule is similar. In particular, we present in the next diagram a critical pair in

which the ($\underline{\delta}$) and the ($\underline{es}$) redexes are not disjoint. We recall that there exists no infinite ($\underline{\delta}$) or ($\{\underline{es}, \underline{ac}\}$) reduction by Lemma 33. For the sake of simplicity, the diagram shows a single ($\underline{\delta}$) step. A longer derivation would just bring to a further duplication of the ($\{\underline{es}, \underline{ac}\}$) redex but the diagram could be closed in a similar way.

$$\begin{array}{ccc}
\mathsf{C}_0\{(G_1 \underline{\wr} G_2)\ \mathsf{C}_1\{\underline{x}\}\}\ [x = G, E] & \xrightarrow[\underline{\delta}]{} & \mathsf{C}_0\{(G_1\ \mathsf{C}_1\{\underline{x}\}) \wr (G_2\ \mathsf{C}_1\{\underline{x}\})\ [x = G, E] \\
\Big\downarrow{\scriptstyle \{\underline{es},\underline{ac}\}} & & \Big\downarrow{\scriptstyle \{\underline{es},\underline{ac}\}} \\
\mathsf{C}_0\{(G_1 \underline{\wr} G_2)\ \mathsf{C}_1\{G\}\}\ [x = G, E] & \xrightarrow[\underline{\delta}]{} & \mathsf{C}_0\{(G_1\ \mathsf{C}_1\{G\}) \wr (G_2\ \mathsf{C}_1\{G\})\ [x = G, E]
\end{array}$$

□

Taking advantage of the previous three lemmata, it is now possible to show the confluence modulo *AC1* of the $\mapsto_{\underline{\Sigma}}$ relation.

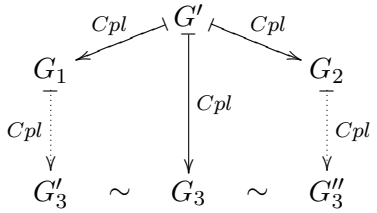
PROPOSITION 39 (CON$_\sim(\underline{\Sigma})$). *The relation $\mapsto_{\underline{\Sigma}}$ is confluent modulo AC1.*

Proof: By Proposition 21, using Lemma 36, Lemma 37 and Lemma 38. □

Using the weak termination of the relation $\mapsto_{\underline{\Sigma}}$ and its confluence modulo *AC1*, we can finally prove that the diamond property modulo *AC1* holds for the *Cpl* relation.

LEMMA 40 (D$_\sim$(*Cpl*)). *The rewrite relation Cpl enjoys the diamond property modulo AC1.*

Proof: Given a term G', let $S = S_1 \cup S_2$ be a set of underlined redexes in G' such that we have $G' \mapsto_{Cpl} G_3$ reducing all the underlined redexes in S. Let G_1 and G_2 be the two partial developments relative to S_1 and S_2 respectively, *i.e.* $G' \mapsto_{Cpl} G_1$ reducing only the redexes in S_1 and $G' \mapsto_{Cpl} G_2$ reducing only the redexes in S_2. In both cases, since $WN(\underline{\Sigma})$ holds by Proposition 34, we can continue reducing the remaining underlined redexes, obtaining $G_1 \mapsto_{Cpl} G_3'$ and $G_2 \mapsto_{Cpl} G_3''$. Since all the steps in the *Cpl* reduction are $\underline{\Sigma}$ steps, using the fact that G_3, G_3' and G_3'' are completely reduced *w.r.t.* $\underline{\Sigma}$ and that $CON_\sim(\underline{\Sigma})$ holds by Proposition 39, we can conclude on the equivalence of G_3, G_3' and G_3''.



□

The confluence of the $\mapsto_{\Sigma}$ relation follows easily by noticing that this relation and the *Cpl* relation have the same transitive closure.

PROPOSITION 41 (CON$_\sim(\Sigma)$). *The relation $\mapsto_{\Sigma}$ is confluent modulo AC1.*

Proof: The result follows by Lemma 40, since if *Cpl* satisfies the diamond property modulo *AC1*, so does its transitive closure and it is not difficult to show that the transitive closure of the relation *Cpl* is the same as that of the relation $\mapsto_{\Sigma}$. This follows from the fact that $\mapsto_{\Sigma} \subseteq \mapsto_{Cpl} \subseteq \mapsto_{\Sigma}$. The first inclusion can be proved by underlining the redex reduced by the Σ step. The second inclusion follows trivially from the definition of the *Cpl* relation. □

General confluence

So far we have shown the confluence of the relations induced by two subsets of rules τ and Σ separately. In the last part of the proof we consider the union of these two subsets of rules. General confluence holds since we can prove the commutation modulo *AC1* between the relation $\mapsto_{\tau}$ and the relation $\mapsto_{\Sigma}$.

LEMMA 42 ($\mathrm{COM}_{\sim}(\tau, \Sigma)$). *The relations* $\mapsto_{\tau}$ *and* Σ *commute modulo AC1.*

Proof: Since the relation $\mapsto_{\Sigma}$ does not terminate, it is easier to show strong commutation between the two relations instead of general commutation.

$$\begin{array}{ccc} G & \xrightarrow[\tau]{} & G_1 \\ {\scriptstyle\Sigma}\downarrow & & {\scriptstyle\Sigma}\,\vdots\,{\scriptstyle 0/1} \\ G_2 & \overset{\tau}{\twoheadrightarrow} & G_1' \sim G_2' \end{array}$$

We can then conclude by Proposition 22, using the compatibility with *AC1* for the relations $\mapsto_{\tau}$ and $\mapsto_{\Sigma}$, which follows from Lemma 24. The possibility of closing the diagram by using at most one step for the Σ-rules is ensured by the fact that none of the τ-rules is duplicating.

If the applied Σ-rule is the (δ) rule, the diagram can be easily closed, since the τ-rules do not interfere with (δ) redexes (the generated critical pairs are trivial). Only the (*garbage*) rule can alter a (δ) redex by eliminating it and in this case the diagram is closed with zero (δ) steps.

If the applied Σ-rule is a substitution rule, we analyse next the interesting critical pairs.

- The τ-rule applied to G is the (*propagate*) rule. The only interesting case is the following where the two Σ-rules applied are different.

$$\begin{array}{ccc} P \ll (\mathsf{Ctx}\{y\}\ [y = H, E]) & \xrightarrow[p]{} & P \ll \mathsf{Ctx}\{y\}, y = H, E \\ {\scriptstyle es}\downarrow & & {\scriptstyle ac}\,\vdots \\ P \ll (\mathsf{Ctx}\{H\}\ [y = H, E]) & \overset{p}{\twoheadrightarrow} & P \ll \mathsf{Ctx}\{H\}, y = H, E \end{array}$$

- The τ-rule applied to G is the (*decompose*) rule. In this case the term G is of the form $H\ [f(H_1, \ldots, H_n) \ll f(\mathsf{Ctx}\{y\}, \ldots, H_n'), y = H', E]$. The (*decompose*) rule transforms the match equation in a set of simpler constraints $H_1 \ll \mathsf{Ctx}\{y\}, \ldots, H_n \ll H_n'$ in the same list. Since the (*acyclic sub*) rule is applied using matching modulo *AC1*, the substitution generated from $y = H'$ can be equivalently performed either before or after the decomposition.

- The τ-rule applied to G is the (*solved*) rule. In this case, there are no differences between G_1 and G from the point of view of the application of a substitution rule.

- The τ-rule applied to G is the (*garbage*) rule. The particularity here is that we can have zero steps of the Σ rules for closing the diagram when the substitution redex is part of the subterm which is eliminated by garbage collection.

- The τ-rule applied to G is the (*black hole*) rule. We may have an overlap of the (*external sub*) rule and the (*black hole*) rule if the term duplicated by the substitution is a variable.

$$\begin{array}{ccc} \mathsf{Ctx}\{y\}\ [y = y, E] & \xrightarrow[bh]{} & \mathsf{Ctx}\{\bullet\}\ [y = y, E] \\ {\scriptstyle es}\downarrow & & {\scriptstyle es}\,\vdots\,{\scriptstyle 0} \\ \mathsf{Ctx}\{y\}\ [y = y, E] & \overset{bh}{\twoheadrightarrow} & \mathsf{Ctx}\{\bullet\}\ [y = y, E] \end{array}$$

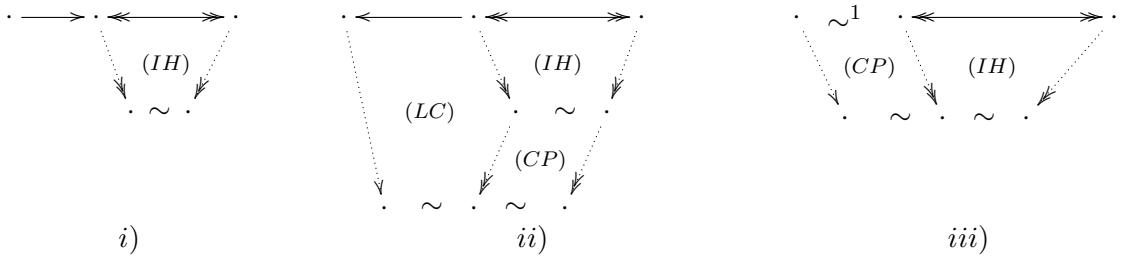


Figure 13. Church-Rosser property for $\rho_g/AC1$.

If the cycle has length greater than one, *i.e.* it is expressed by more than one recursion equation, the matching modulo *AC1* allows to apply the (*black hole*) rule even when the recursion equations are not in the right order in the list and this can happen as a consequence of the application of the (*external sub*) rule.

$$\begin{array}{ccc} \mathsf{Ctx}\{y\}\ [y=x, x=y, E] & \xrightarrow[bh]{} & \mathsf{Ctx}\{\bullet\}\ [y=x, x=y, E] \\ \Big\downarrow{\scriptstyle es} & & \Big\downarrow{\scriptstyle es}\ 0 \\ \mathsf{Ctx}\{x\}\ [y=x, x=y, E] & \xrightarrow[bh]{} & \mathsf{Ctx}\{\bullet\}\ [y=x, x=y, E] \end{array}$$

We have similar cases for the (*acyclic sub*) rule.

□

The confluence modulo *AC1* of the sets of rules τ and Σ, the commutation modulo *AC1* of the two sets, together with their compatibility property with *AC1* ensure the confluence of their union.

THEOREM 43 ($\mathrm{CON}_\sim(\rho_g, AC1)$). *In the linear ρ_g-calculus, the rewrite relation $\mapsto\!\!\!\!\!\rightarrow_{\rho_g, AC1}$ is confluent modulo AC1.*

Proof: By Proposition 21 using Proposition 29, Proposition 41, Lemma 24 and Lemma 42. □

As mentioned in the first section, what we aim at is a more general result about rewriting on *AC1*-equivalence classes of ρ_g-graphs. Thanks to the property of compatibility of ρ_g with *AC1*, the Church-Rosser property on *AC1*-equivalence classes for the ρ_g-calculus rewrite relation can be easily derived from the latter theorem.

THEOREM 44 ($CR_\sim(\rho_g/AC1)$). *The linear ρ_g-calculus is Church-Rosser modulo AC1.*

Proof: By induction on the length of the reduction.
To emphasise the first step, we decompose the reduction $\twoheadleftarrow\!\!\!\!\!\rightarrow^n_{\rho_g/AC1}$ into $\twoheadleftarrow\!\!\!\!\!\rightarrow^1_{\rho_g/AC1}\ \twoheadleftarrow\!\!\!\!\!\rightarrow^{n-1}_{\rho_g/AC1}$.
We have three possibilities for the first step. For each case we show the Church-Rosser diagram in Figure 13, where *LC* stands for local confluence modulo *AC1*, *CP* stands for compatibility with *AC1* and *IH* stands for induction hypothesis. □

4. Expressiveness of the ρ_{g}-calculus

4.1. ρ_{g}-CALCULUS VERSUS ρ-CALCULUS AND CYCLIC λ-CALCULUS

The set of terms of the ρ-calculus is a strict subset of the set of ρ_{g}-graphs of the ρ_{g}-calculus (modulo some syntactic conventions). The main difference for ρ-terms is the restriction of the list of constraints to a single constraint necessarily of the form $\_ \ll \_$ (delayed matching constraint).

Before proving that the ρ-calculus is simulated in the ρ_{g}-calculus, we need to show that the MATCHING RULES of the ρ_{g}-calculus are well-behaved with respect to the ρ-calculus matching algorithm restricted to patterns (Cirstea et al., 2002).

LEMMA 45. *Let T be an algebraic ρ-term with $\mathcal{FV}(T) = \{x_1, \ldots, x_n\}$ and let $T \ll U$ be a matching problem with solution $\sigma = \{x_1/U_1, \ldots, x_n/U_n\}$, i.e. $\sigma(T) = U$. Then we have $T \ll U \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{M}} x_1 = U_1, \ldots, x_n = U_n$.*

Proof: We show by structural induction on the term T that there exists a reduction of the form $T \ll U \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{M}} x_1 \ll U_1, \ldots, x_n \ll U_n$, where the x_i's are all distinct and thus the thesis follows.

- *Basic case*: The term T is a variable or a constant. The case where $T = x$ is trivial. If $T = a$ then $\sigma = \{\}$ and $U = a$. In the ρ_{g}-calculus we have $a \ll a \mapsto_e \epsilon$ and the property obviously holds.
- *Induction case*: $T = f(T_1, \ldots, T_m)$ with $m > 0$.
  Since a substitution σ exists and the matching is syntactic, we have $U = f(V_1, \ldots, V_m)$ and $\sigma(f(T_1, \ldots, T_m)) = f(\sigma(T_1), \ldots, \sigma(T_m))$ with $\sigma(T_i) = V_i$, for $i = 1 \ldots m$. By induction hypothesis, for any i, if $\mathcal{FV}(T_i) = \{x_1^i, \ldots, x_{k_i}^i\} \subseteq \mathcal{FV}(T)$, then we have the reduction $T_i \ll V_i \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{M}} x_1^i \ll \sigma(x_1^i), \ldots, x_{k_i}^i \ll \sigma(x_{k_i}^i)$. Joining the various reductions we have $f(T_1, \ldots, T_m) \ll f(V_1, \ldots, V_m) \mapsto_{dk} T_1 \ll V_1, \ldots, T_m \ll V_m \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{M}} x_1 \ll \sigma(x_1), \ldots, x_n \ll \sigma(x_n)$. To understand the last step note that in the list

  $$x_1^1 \ll \sigma(x_1^1), \ldots, x_{k_1}^1 \ll \sigma(x_{k_1}^1), \ldots, x_1^m \ll \sigma(x_m^1), \ldots, x_{k_m}^m \ll \sigma(x_{k_m}^m)$$

  constraints with the same left-hand side variable have identical right-hand sides. Hence, by idempotency, such list coincides with $x_1 \ll \sigma(x_1), \ldots, x_n \ll \sigma(x_n)$.

□

We can show now that a reduction in the ρ-calculus can be simulated in the ρ_{g}-calculus.

LEMMA 46. *Let T and T' be ρ-terms. If there exists a reduction $T \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta} T'$ in the ρ-calculus then there exists a corresponding one $T \mapsto\!\!\!\!\twoheadrightarrow_{\mathsf{g}} T'$ in the ρ_{g}-calculus.*

Proof: We show that for each reduction step in the ρ-calculus we have a corresponding sequence of reduction steps in the ρ_{g}-calculus.

- If $T \mapsto_\rho T'$ or $T \mapsto_\delta T'$ in the ρ-calculus, then we trivially have the same reduction in the ρ_{g}-calculus using the corresponding rules.
- If $T = [T_1 \ll T_3]T_2 \mapsto_\sigma \sigma(T_2) = T'$ where T_1 is a ρ-calculus pattern and the substitution $\sigma = \{U_1/x_1, \ldots, U_m/x_m\}$ is solution of the matching then, in the ρ_{g}-calculus the corresponding reduction is the following:

  $$\begin{array}{ll} T = & T_2\ [T_1 \ll T_3] \\ \mapsto\!\!\!\!\twoheadrightarrow_{\mathcal{M}} & T_2\ [x_1 = U_1, \ldots, x_m = U_m] \quad (\textit{by Lemma 45}) \\ \mapsto\!\!\!\!\twoheadrightarrow_{es} & T'\ [x_1 = U_1, \ldots, x_m = U_m] \\ \mapsto\!\!\!\!\twoheadrightarrow_{gc} & T'\ [\epsilon] \\ \mapsto_{gc} & T' \end{array}$$

□

THEOREM 47 (Completeness). *Let T and T' be ρ-terms. If there exists a reduction $T \mapsto\!\!\!\!\!\rightarrow_{\rho\sigma\delta} T'$ in the ρ-calculus then $T \mapsto\!\!\!\!\!\rightarrow_{\rho g} T'$ in the ρ_g-calculus.*

Proof: Follows from Lemma 46. □

In the case of matching failures, the two calculi handle errors in a slightly different way, even if, in both cases, matching clashes are not reduced and kept as constraint application failures. In particular we can have a deeper decomposition of a matching problem in the ρ_g-calculus than in the ρ-calculus and thus it can happen that a ρ-term in normal form can be further reduced in the ρ_g-calculus.

EXAMPLE 48 (Matching failure in ρ-calculus and ρ_g-calculus). *In both calculi, non successful reductions lead to a non solvable match equation in the list of constraints of the term.*

$$\begin{array}{ll} & (f(a) \twoheadrightarrow b)\ f(c) \\ \mapsto_{\rho\sigma\delta} & [f(a) \ll f(c)]b \end{array} \qquad \begin{array}{ll} & (f(a) \twoheadrightarrow b)\ f(c) \\ \mapsto_{\rho} & b\ [f(a) \ll f(c)] \\ \mapsto_{dk} & b\ [a \ll c] \end{array}$$

Notice that in the ρ-calculus, since the matching algorithm cannot compute a substitution solving the match equation $f(a) \ll f(c)$, the (σ) rule cannot be applied and thus the reduction is stuck. On the other hand, in the ρ_g-calculus the MATCHING RULES *can partially decompose the match equation until the clash $a \ll c$ is reached.*

The terms of $\lambda\phi_0$ can be easily translated into terms of the ρ_g-calculus. The main difference of $\lambda\phi_0$ *w.r.t.* the ρ_g-calculus is the restriction of the list of constraints to a list of recursion equations. Delayed matching constraints are not needed since in the λ-calculus the matching is always trivially satisfied.

DEFINITION 49 (Translation). *The translation of a $\lambda\phi_0$-term t into a ρ_g-term, denoted $\overline{t}$, is inductively defined as follows:*

$$\begin{array}{lcl} \overline{x} & \triangleq & x \\ \overline{\lambda x.t} & \triangleq & x \twoheadrightarrow \overline{t} \\ \overline{t_0\ t_1} & \triangleq & \overline{t_0}\ \overline{t_1} \end{array} \qquad \begin{array}{lcl} \overline{f(t_1,\ldots,t_n)} & \triangleq & f(\overline{t_1},\ldots,\overline{t_n}) \\ \overline{\langle t_0|\ x_1 = t_1,\ldots,x_n = t_n\rangle} & \triangleq & \overline{t_0}\ [x_1 = \overline{t_1},\ldots,x_n = \overline{t_n}] \end{array}$$

We can see the evaluation rules of the ρ_g-calculus as the generalisation of those of the $\lambda\phi_0$-calculus. The *β-rule* can be simulated using the BASIC RULES of the ρ_g-calculus. The rest of the rules can be simulated using the corresponding ones in the subset GRAPH RULES of the ρ_g-calculus.

We show next that a reduction in the $\lambda\phi_0$-calculus can be simulated in the ρ_g-calculus.

LEMMA 50. *Let t_1 and t_2 be two $\lambda\phi_0$-terms. If $t_1 \mapsto_{\lambda\phi} t_2$ in the cyclic λ-calculus, then there exists a reduction $\overline{t_1} \mapsto\!\!\!\!\!\rightarrow_{\rho g} \overline{t_2}$ in the ρ_g-calculus.*

Proof: We proceed by analysing each reduction axiom of $\lambda\phi_0$.

- β-rule: $t_1 = (\lambda x.s_1)\ s_2 \to_\beta \langle s_1|\ x = s_2\rangle = t_2$

  In the ρ_g-calculus we have:

$$\overline{t_1} \ = \ (x \twoheadrightarrow \overline{s_1})\ \overline{s_2} \ \mapsto_\rho \ \overline{s_1}\ [x \ll \overline{s_2}] \ \mapsto_s \ \overline{s_1}\ [x = \overline{s_2}] \ = \ \overline{t_2}$$

- *external sub* rule: trivial.
- *acyclic sub* rule: trivial ($\lll$ stands always for $=$ in this case).

– *black hole* rule: trivial.

– *garbage collect* rule: The proviso $E \perp (E', t)$ is equivalent to the one expressed using the definition of free variables in the ρ_{g}-calculus. The condition $E' \neq \epsilon$ is implicit in the ρ_{g}-calculus since we eliminate one recursion equation at time. For this reason, a single step of the *garbage collect* rule in $\lambda\phi_0$ can correspond to several steps of the corresponding *garbage* rule in the ρ_{g}-calculus: if $\langle t|E, E'\rangle \to_{gc} \langle t|E\rangle$ then $\overline{t}\ [E, E'] \mapsto\!\!\!\!\to_{gc} \overline{t}\ [E]$.

□

THEOREM 51 (Completeness). *Let t_1 and t_2 be two $\lambda\phi$-terms. Given a reduction $t_1 \mapsto\!\!\!\!\to_{\lambda\phi} t_2$ in the cyclic λ-calculus, then there exists a corresponding reduction $\overline{t_1} \mapsto\!\!\!\!\to_{\mathsf{g}} \overline{t_2}$ in the ρ_{g}-calculus.*

Proof: Follows from Lemma 50. □

4.2. Simulation of term graph rewriting into the ρ_{g}-calculus

The possibility of representing structures with cycles and sharing naturally leads to the question asking whether first-order term graph rewriting can be simulated in this context. In this section we provide a positive answer. For our purposes, we choose the equational description of term graph rewriting defined in Section 1.3. We recall that a term graph rewrite system $TGR = (\Sigma, \mathcal{R})$ is composed by a signature Σ over which the considered term graphs are built and a set of term graph rewrite rules $\mathcal{R}$. Both the term graphs over Σ and the set of rules are translated at the object level of the ρ_{g}-calculus, *i.e.* into ρ_{g}-graphs.

DEFINITION 52. *We define for the various components of a TGR the corresponding element in the ρ_{g}-calculus.*

– (Terms) *Using the equational framework, the set of term graphs of a TGR is a strict subset of the set of terms of the ρ_{g}-calculus, modulo some obvious syntactic conventions. In particular, by abuse of notation, in the following we will sometimes confuse the two notations $\{x \mid E\}$ and $x\ [E]$.*

– (Rewrite rules) *A rewrite rule $(L, R) \in \mathcal{R}$ is translated into the corresponding ρ_{g}-graph $L \twoheadrightarrow R$. Recall that we consider only left-linear term graph rewrite rules.*

– (Substitution) *A substitution $\sigma = \{x_1/G_1, \ldots x_n/G_n\}$ corresponds in the ρ_{g}-calculus to a list of constraints $E = (x_1 = G_1, \ldots, x_n = G_n)$ and its application to a term graph L corresponds to the addition of the list of constraints to the ρ_{g}-term L, i.e. to the ρ_{g}-graph $L\ [E]$.*

As seen in Section 1.3, it is convenient to work with a restricted class of term graphs in flat form and without useless equations. The structure of a ρ_{g}-graph can be, in general, more complex than the one of a flat term graph, since it can have nested lists of constraints and garbage. To recover the similarity, we define next the canonical form of a ρ_{g}-graph G containing no abstractions and no match equations.

DEFINITION 53 (Canonical form). *Let G be a ρ_{g}-graph containing no abstractions and no match equations. We say that G is in canonical form if it is in flat form and it contains neither garbage equations nor trivial equations of the form $x = y$.*

Any ρ_{g}-graph G without abstractions and match equations can be transformed into a corresponding ρ_{g}-graph in canonical form, that will be denoted by $\overline{G}$, as follows. We first perform the flattening and merging of the lists of equations of G and we introduce new recursion equations

with fresh variables for every subterm of G. We obtain in this way a $\rho_{\mathtt{g}}$-graph in flat form, where the notion of flat form is defined analogously to term-graphs. For instance, the $\rho_{\mathtt{g}}$-term $x\ [x = f(g(y))\ [y = z,\ z = a]]$ would become $x\ [x = f(w),\ w = g(y), y = z,\ z = a]$. The canonical form can then be obtained from the flat form by removing the useless equations, by means of the two substitution rules and the garbage collection rule of the $\rho_{\mathtt{g}}$-calculus. In the example above we would get $x\ [x = f(w),\ w = g(z),\ \ z = a]$. It is easy to see that the canonical form of a $\rho_{\mathtt{g}}$-graph is unique, up to α-conversion and the $AC1$ axioms for the constraint conjunction operator, and a $\rho_{\mathtt{g}}$-graph with no abstractions and no match equations in canonical form can be seen as a term graph in flat form.

Before proving the correspondence between rewritings in a TGR and in the $\rho_{\mathtt{g}}$-calculus, we need a lemma showing that matching in the $\rho_{\mathtt{g}}$-calculus is well-behaved *w.r.t.* the notion of term graph homomorphism.

LEMMA 54 (Matching). *Let G be a closed term graph and let (L, R) be a left-linear rewrite rule, with $\mathcal{V}ar(L) = \{x_1, \ldots, x_m\}$. Assume that there is an homomorphism from L to G, given by the variable renaming $\sigma = \{x_1/x'_1, \ldots, x_m/x'_m\}$.*

Let $E = (x_n = x'_n, \ldots, x_m = x'_m, E_G)$ with $\{x_n, \ldots, x_m\} = \mathcal{FV}(L)$. Then in the $\rho_{\mathtt{g}}$-calculus we have the reduction $L \ll G \mapsto\!\!\!\!\!\rightarrow_{\not{\mathtt{g}}} E$ with $\tau(\overline{L\ [E]}) = G$, where τ is a variable renaming.

Proof: We consider functions of arity less or equal two. Note that this is not really a restriction since n-ary functions are encoded in the $\rho_{\mathtt{g}}$-calculus as a sequence of nested binary applications.

Given the matching problem $L \ll G$, where $L = x_1\ [E_L]$ and $G = x'_1\ [E_G]$, in the $\rho_{\mathtt{g}}$-calculus we have the reduction

$$L \ll G\ =\ x_1\ [E_L] \ll x'_1\ [E_G]\ \mapsto_p\ x_1\ [E_L] \ll x'_1, E_G\ \mapsto\!\!\!\!\!\rightarrow_{es,gc}\ T_L \ll x'_1, E_G$$

where T_L is a term without constraints, *i.e.* a tree, which can be reached since L is linear and acyclic by hypothesis.

We proceed by induction on the length of the list of recursion equations E_L of the term graph L, or, equivalently, on the height of T_L, seen as tree.

Base case. T_L is a variable x_1. We obtain the reduction $x_1 \ll x'_1, E_G\ \mapsto\!\!\!\!\!\rightarrow_{\mathtt{s}}\ x_1 = x'_1, E_G$.

Then it is immediate to verify that $\overline{L\ [E]} = \overline{x_1\ [x_1 = x'_1, E_G]}$ is equal to G using the variable renaming $\tau = \{x_1/x'_1\}$.

Induction. Let G be of the form $G = x'_1\ [x'_1 = f(x'_2, x'_3), x'_2 = T_2, x'_3 = T_3, E']$. Continuing the reduction of the match equation $L \ll G$ we obtain

$$\begin{array}{rcl} T_L \ll x'_1, E_G & = & T_L \ll x'_1, x'_1 = f(x'_2, x'_3), \ldots \\ & \mapsto_{ac} & T_L \ll f(x'_2, x'_3), E_G \end{array}$$

Since by hypothesis an homomorphism σ between L and G exists, we have $T_L = f(T'_2, T'_3)$ and thus

$$T_L \ll f(x'_2, x'_3), E_G \mapsto_{dk} T'_2 \ll x'_2, T'_3 \ll x'_3, E_G$$

Using the induction hypothesis and the fact that L is acyclic, we obtain the reductions $T'_2 \ll x'_2, E_G \mapsto\!\!\!\!\!\rightarrow_{\not{\mathtt{g}}} E_2$ and $T'_3 \ll x'_3, E_G \mapsto\!\!\!\!\!\rightarrow_{\not{\mathtt{g}}} E_3$ and the variable renamings τ_2 and τ_3 such that $\tau_2(\overline{T'_2\ [E_2]}) = T_2$ and $\tau_3(\overline{T'_3\ [E_3]}) = T_3$. Therefore, since $\mathcal{FV}(L) = \mathcal{FV}(T'_2) \cup \mathcal{FV}(T'_3)$, it is easy to verify that $L \ll G \mapsto\!\!\!\!\!\rightarrow_{\not{\mathtt{g}}} E$, with $E = E_2, E_3, E_G$ and that the variable renaming $\tau = \tau_2\tau_3\{x_1/x'_1\}$ is such that $\tau(\overline{L\ [E]}) = G$. □

The previous lemma guarantees the fact that if there exists an homomorphism (represented as a variable renaming) of a term graph L into G, in the $\rho_{\mathtt{g}}$-calculus we obtain the variable renaming (in the form of a list of recursion equations) as result of the evaluation of the matching problem $L \ll G$. In other words, this means that if a rewrite rule can be applied to a term graph, the application is still possible after the translation of the rule into $\rho_{\mathtt{g}}$-abstraction and of the term graph into a $\rho_{\mathtt{g}}$-graph.

rewrite rule: $L \to R$ rewrite step: $G \to G'$

Figure 14. Example of rewriting in a TGR.

EXAMPLE 55 (Matching). *Consider the term graphs* $L = \{x_1 \mid x_1 = add(x_2, y_2), x_2 = s(y_1)\}$ *and* $G = \{z_0 \mid z_0 = add(z_1, z_2),\ z_1 = s(z_2), z_2 = 0\}$ *(see Figure 14) and the homomorphism* $\sigma = \{x_1/z_0,\ x_2/z_1,\ y_1/z_2,\ y_2/z_2\}$ *from* L *to* G*. We show how* σ *can be obtained in the* ρ_{g}*-calculus starting from the matching problem* $L \ll G$*.*

$$
\begin{array}{lll}
L \ll G & \mapsto_{p} & L \ll z_0, E_G \\
& \mapsto\!\!\!\!\to_{es,gc} & add(s(y_2), y_1) \ll z_0, E_G \\
& = & add(s(y_2), y_1) \ll z_0, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
& \mapsto_{ac} & add(s(y_2), y_1) \ll add(z_1, z_2), z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
& \mapsto_{dk} & s(y_2) \ll z_1, y_1 \ll z_2, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
& \mapsto\!\!\!\!\to_{ac,dk} & y_2 \ll z_2, y_1 \ll z_2, z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0 \\
& \mapsto\!\!\!\!\to_{s} & y_2 = z_2, y_1 = z_2, E_G
\end{array}
$$

We can verify then that $\overline{L\ [y_2 = z_2, y_1 = z_2, E_G]}$ *is equal to* G *up to variable renaming. In fact, the transformation into the canonical form leads to the graph* $x_1\ [x_1 = add(x_2, z_2), x_2 = s(z_2), z_2 = 0]$ *and it is easy to see that the variable renaming* $\tau = \{x_1/z_0, x_2/z_1\}$ *makes this graph equal to* G*.*

We analyse next the relationship between derivations of a term graph rewrite system $TGR = (\Sigma, \mathcal{R})$ and reductions in the ρ_{g}-calculus. Given a term graph G in the TGR and a derivation *w.r.t.* the set of rules $\mathcal{R}$, we show how to build a ρ_{g}-graph which reduces in the ρ_{g}-calculus to a term corresponding to the ending term graph of the original TGR reduction.

Notice that, since in the ρ_{g}-calculus the rule application is at the object level, we need to define a *position trace* ρ_{g}*-graph* encoding the position of the redex in the given term graph G. For doing this, we use an annotated path that leads to this redex position in G.

DEFINITION 56 (Position trace term). *Let* $G = y_0\ [y_0 = f_1(y_1, \ldots y_n), E]$ *be a term graph. We define* $G \downarrow_j = y_j\ [E_G]$*, where* $j = 0, \ldots, n$*. Given an annotated path* $p = f_1 i_1 \ldots i_{m-1} f_m$ *in* G *and a set of fresh variables* $x_0, x_1^j, \ldots, x_n^j$ *for* $j = 1, \ldots, m-1$ *and* $n \in \mathbb{N}$*, we recursively define the* position trace ρ_{g}-graph $\mathcal{P}_p(G)$ *as*

$$\mathcal{P}_{f_j i_j p'}(G) = f_j(x_1^j, \ldots, \mathcal{P}_{p'}(G \downarrow_j), \ldots, x_n^j) \quad \textit{and} \quad \mathcal{P}_{f_m}(G) = x_0$$

where f_j *is of arity* n *and has* $\mathcal{P}_{p'}(G \downarrow_j)$ *as* i_j*-th argument.*

We obtain thus a ρ_{g}-graph $\mathcal{P}_p(G)$ which has the same structure of G and whose positions are filled using the information given by the annotated path in G, if any, otherwise using fresh variables. It can be easily seen that, by construction, there is an homomorphism from the position trace term $\mathcal{P}_p(G)$ into G. The position trace graph is then used to build a ρ_{g}-graph H that pushes the rewrite rule down to the right application position, according to the given term graph rewrite step.

LEMMA 57 (Simulation). *Let* G *be a term graph, let* (L, R) *be a left-linear rewrite rule rooted at* z *and let* σ *be an homomorphism from* L *to* G *such that* $G_{\lceil \sigma(z) = t \rceil_p} \to G_{[\sigma(R)]_p}$*.*

Define the ρ_{g}*-term* $H = \mathcal{P}_p(G)_{\lceil x \rceil_p} \twoheadrightarrow \mathcal{P}_p(G)_{[(L \twoheadrightarrow R)\ x]_p}$*. Then in the* ρ_{g}*-calculus there exists a reduction* $(H\ G) \mapsto\!\!\!\!\to_{\rho\mathsf{g}} G'$ *and a variable renaming* τ *such that* $\tau(\overline{G'})$ *is equal to* $G_{[\sigma(R)]_p}$*.*

Proof: First of all, observe that by definition $\mathcal{P}_p(G)_{\lceil z \rceil_p}$ matches G. If the corresponding homomorphism is $\sigma' = \{z/z', z_1/z'_1, \ldots, z_k/z'_k, \ldots, z_n/z'_n\}$, by Lemma 54 we obtain as solution of the matching in the ρ_{g}-calculus a list of recursion equations $z = z', z_1 = z'_1, \ldots, z_k = z'_k, E_G$ where $\{z, z_1, \ldots, z_k\} = \mathcal{FV}(\mathcal{P}_p(G))$. When we do not need to identify each single equation, we will denote such list simply by E'_G. Hence $\mathcal{P}_p(G)_{\lceil z \rceil_p}\ [E'_G]$ is equal up to variable renaming to $G_{\lceil z' \rceil_p}$ and similarly, $\overline{\mathcal{P}_p(G)_{\lceil T \rceil_p}\ [E'_G]}$ is equal up to variable renaming to $\overline{G_{\lceil T \rceil_p}}$ for any term T.

In the ρ_{g}-calculus we obtain the following reduction:

$$
\begin{array}{ll}
 & \mathcal{P}_p(G)_{\lceil z \rceil_p} \twoheadrightarrow \mathcal{P}_p(G)_{[(L \twoheadrightarrow R)\ z]_p}\ G \\
\mapsto_{\rho} & \mathcal{P}_p(G)_{[(L \twoheadrightarrow R)\ z]_p}\ [\mathcal{P}_p(G)_{\lceil z \rceil_p} \ll G] \\
\mapsto\!\!\!\!\twoheadrightarrow_{\mathsf{fg}} & \mathcal{P}_p(G)_{[(L \twoheadrightarrow R)\ z]_p}\ [z = z', \ldots, z_k = z'_k, E_G] \quad \textit{by Lemma 54} \\
\mapsto_{es} & \mathcal{P}_p(G)_{[(L \twoheadrightarrow R)\ z']_p}\ [E'_G] \\
\mapsto_{\rho} & \mathcal{P}_p(G)_{[R\ [L \ll z']]_p}\ [E'_G] \\
\mapsto_{\mathsf{fg}} & \mathcal{P}_p(G)_{[R\ [y_1 = y'_1, \ldots, y_n = y'_n]]_p}\ [E'_G] \quad \textit{by Lemma 54} \\
\mapsto\!\!\!\!\twoheadrightarrow_{es,gc} & \mathcal{P}_p(G)_{[R']_p}\ [E'_G] \ =\ G'_1
\end{array}
$$

where $\{y_1, \ldots, y_n\} = \mathcal{FV}(R)$ and R' is the term obtained from R by renaming y_i with y'_i, for $i = 1 \ldots n$. By using Lemma 54, it is not difficult to deduce that R' is equal to $\sigma(R)$ modulo α-conversion. We conclude by noticing that (the flat form of) $\mathcal{P}_p(G)_{\lceil \sigma(R) \rceil_p}\ [E'_G]$ equal up to variable renaming to $G_{[\sigma(R)]_p}$. □

Notice that in the ρ_{g}-graph H we could have separated the rule from the information about its application position by choosing $H = y \twoheadrightarrow (\mathcal{P}_p(G)_{\lceil x \rceil_p} \twoheadrightarrow \mathcal{P}_p(G)_{[y\ x]_p})$ and then by considering $H\ (L \twoheadrightarrow R)\ G$ as starting term of the reduction.

We point out that, in the proof of the previous lemma, if we started with a ρ_{g}-graph equal up to variable renaming to G, say G', we could have constructed an analogous reduction in the ρ_{g}-calculus. Indeed in this case using the same reasoning as above, we obtain as final term $G'_1 = \mathcal{P}_p(G')_{[R']_p}\ [E_G]$ whose flat form is equal up to variable renaming to $G'_{[\sigma(R)]_p}$ which is in turn equal up to variable renaming to $G_{[\sigma(R)]_p}$ and thus the lemma still holds.

COROLLARY 58. *Let G be a term graph and let (L, R) be a left-linear rewrite rule such that $G_{\lceil \sigma(z) = t \rceil_p} \to G_{\lceil \sigma(R) \rceil_p} = G_1$. Then we can construct ρ_{g}-graph H such that for any term G' (whose flat form is) equal up to variable renaming to G there exists a reduction $(H\ G') \mapsto\!\!\!\!\twoheadrightarrow_{\mathsf{fg}} G'_1$ such that (the flat form of) G'_1 is equal up to variable renaming to G_1.*

The final ρ_{g}-graph that we obtain is not exactly the same as the term graph resulting from the ρ_{g}-reduction in the TGR, and this is due to some unsharing steps that may occur in the reduction. In general, the two graphs are equal up to variable renaming, meaning that the ρ_{g}-graph G'_1 is possibly more "unravelled" than the term graph G_1.

EXAMPLE 59 (Addition). *Let (L, R), where $L = x_1\{x_1 = add(x_2, y_1),\ x_2 = s(y_2)\}$ and $R = x_1\{x_1 = s(x_2),\ x_2 = add(y_1, y_2)\}$, be a term graph rewrite rule describing the addition of natural numbers. We apply this rule to the term graph $G = z\{z = s(z_0), z_0 = add(z_1, z_2), z_1 = s(z_2), z_2 = 0\}$ using the variable renaming $\sigma = \{x_1/z_0, x_2/z_1, y_1/z_2, y_2/z_2\}$. We obtain thus the term graph $G' = z\{z = s(z_0), z_0 = s(z'_1), z'_1 = add(z_2, z_2), z_2 = 0\}$. For a graphical representation see Figure 14.*

The corresponding reduction in the ρ_{g}-calculus is as follows. First of all, since the rule is not applied at the head position of G, we need to define the ρ_{g}-graph $H = s(x) \twoheadrightarrow s((L \twoheadrightarrow R)\ x)$ that pushes down the rewrite rule to the right application position, i.e. under the symbol s. Then applying the ρ_{g}-graph H to G we obtain the following reduction:

$$
\begin{array}{ll}
 & s(x) \twoheadrightarrow s((L \twoheadrightarrow R)\ x)\ \ G \\
\mapsto_{\rho} & s((L \twoheadrightarrow R)\ x)\ [s(x) \ll G] \\
\mapsto_{p} & s((L \twoheadrightarrow R)\ x)\ [s(x) \ll z, E_G] \\
\mapsto\!\!\!\!\!\rightarrow_{\not{\mathsf{gc}}} & s((L \twoheadrightarrow R)\ x)\ [x = z_0, E_G] \\
\mapsto_{es,gc} & s((L \twoheadrightarrow R)\ z_0)\ [E_G] \\
\mapsto_{\rho} & s(R\ [L \ll z_0])\ [E_G] \\
\mapsto\!\!\!\!\!\rightarrow_{\not{\mathsf{gc}}} & s(R\ [y_1 = z_2, y_2 = z_2])\ [E_G] \\
= & s(x_1\ [x_1 = s(x_2), x_2 = add(y_1, y_2)]\ [y_1 = z_2, y_2 = z_2])\ [E_G] \\
\mapsto\!\!\!\!\!\rightarrow_{\not{\mathsf{gc}}} & s(x_1\ [x_1 = s(x_2), x_2 = add(z_1, z_2)])\ [E_G] = G''
\end{array}
$$

The canonical form of G'' is then obtained removing the useless recursion equations in E_G and merging the lists of constraints. We get the graph $\overline{G''} = x\ [x = s(x_1), x_1 = s(x_2), x_2 = add(z_1, z_2), z_0 = 0]$ which is equal up to variable renaming to the term graph G'.

THEOREM 60 (Completeness). *Given a n step reduction $G \mapsto\!\!\!\!\rightarrow^n G_n$ in a TGR, then there exist n ρ_{g}-graphs $H_1, \ldots, H_n$ such that $(H_n\ \ldots\ (H_1\ G)) \mapsto\!\!\!\!\!\rightarrow_{\not{\mathsf{gc}}} G'_n$ and there exists a variable renaming τ such that $\tau(\overline{G'_n}) = G_n$.*

Proof: By induction on the length of the reduction, using Lemma 57 and Corollary 58. □

5. Conclusions and future work

In this paper we have proposed the ρ_{g}-calculus, an extension of the ρ-calculus able to deal with graph like structures, where sharing of subterms and cycles (which can be used to represent regular infinite data structures) can be expressed.

The ρ_{g}-calculus is shown to be confluent under suitable linearity assumption over the considered patterns. The confluence result is obtained adapting some techniques for confluence of term rewrite systems to the case of terms with constraints. An additional complication is represented by the fact that ρ_{g}-graphs are considered modulo an equational theory and thus the rewriting relation formally acts on equivalence classes of terms. Since the ρ_{g}-calculus rewrite relation is not terminating, the "finite development method" of the classical λ-calculus together with several properties of "rewriting modulo a set of equations" are needed to obtain the final result, making thus the complete proof quite elaborated.

The ρ_{g}-calculus has been shown to be a quite expressive calculus, able to simulate standard ρ-calculus as well as cyclic λ-calculus and term graph rewriting. The main difference between the ρ_{g}-calculus and TGR lies in the fact that rewrite rules and their control (application position) are defined at the object-level of the ρ_{g}-calculus while in the TGR the reduction strategy is left implicit. The possibility of controlling the application of rewrite rules is particularly useful when the rewrite system is not terminating. It would be certainly interesting to define in the ρ_{g}-calculus iteration strategies and strategies for the generic traversal of ρ_{g}-graphs in order to simulate TGR rewritings guided by a given reduction strategy. A similar work has already been done for representing first-order term rewriting reductions in a typed version of the ρ-calculus (Cirstea et al., 2003). Intuitively, the ρ-term encoding a first-order rewrite systems is a ρ-structure consisting of the corresponding term rewrite rules wrapped in an iterator that allows for the repetitive application of the rules. We conjecture that this approach can be adapted and generalised for handling term-graphs and simulate term-graphs reductions.

At the same time, an appealing problem is the generalisation of ρ_{g}-calculus to deal with different, possibly non syntactic, matching theories. General cyclic matching, namely matching involving cyclic left-hand sides, could be useful, for example, for the modelling of reactions on cyclic molecules or transformations on distribution nets. One should notice that this extension is not straightforward, since, in ρ_{g}-calculus matching is internalised rather than being carried out at metalevel.

Furthermore, a term of the ρ_{g}-calculus, possibly with sharing and cycles, can be seen as a "compact" representation of a possibly infinite ρ-calculus term, obtained by "unravelling" the original term. On the one hand, it would be interesting to define an infinitary version of the ρ-calculus, taking inspiration, e.g., from the work on the infinitary λ-calculus (Kennaway et al., 1997) and on infinitary rewriting (Kennaway et al., 1991; Corradini, 1993). On the other hand, to enforce the view of the ρ_{g}-calculus as efficient implementation of terms and rewriting in the infinitary ρ-calculus one should have an adequacy result in the style of (Kennaway et al., 1994; Corradini and Drewes, 1997).

References

Abadi, M., L. Cardelli, P.-L. Curien, and J.-J. Levy: 1991, 'Explicit Substitutions'. *Journal of Functional Programming* **4**(1), 375–416.

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

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

Barendregt, H.: 1984, *The Lambda-Calculus, its syntax and semantics*, Studies in Logic and the Foundation of Mathematics. Amsterdam: Elsevier Science Publishers B. V. (North-Holland). Second edition.

Barendregt, H. P., M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep: 1987, 'Term Graph Rewriting'. In: *Proceedings of PARLE'87, Parallel Architectures and Languages Europe*, Vol. 259 of *Lecture Notes in Computer Science*. Eindhoven, pp. 141–158, Springer-Verlag.

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

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

Church, A.: 1941, 'A Formulation of the Simple Theory of Types'. *Journal of Symbolic Logic* **5**, 56–68.

Cirstea, H., G. Faure, and C. Kirchner: 2004, 'A rho-calculus of explicit constraint application'. In: *Workshop on Rewriting Logic and Applications*. Electronic Notes in Theoretical Computer Science.

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

Cirstea, H., C. Kirchner, and L. Liquori: 2001, 'Matching Power'. In: A. Middeldorp (ed.): *Proceedings of RTA'01, Rewriting Techniques and Applications*, Vol. 2051 of *Lecture Notes in Computer Science*. Utrecht, The Netherlands, pp. 77–92, Springer-Verlag.

Cirstea, H., C. Kirchner, and L. Liquori: 2002, 'Rewriting Calculus with(out) Types'. In: F. Gadducci and U. Montanari (eds.): *Proceedings of the fourth workshop on rewriting logic and applications*. Pisa (Italy), Electronic Notes in Theoretical Computer Science.

Cirstea, H., L. Liquori, and B. Wack: 2003, 'Rewriting Calculus with Fixpoints: Untyped and First-order Systems'. In: S. Berardi, M. Coppo, and F. Damian (eds.): *Types for Proofs and Programs (TYPES)*, Vol. 3085 of *Lecture Notes in Computer Science*. Torino (Italy), pp. 147–171.

Corradini, A.: 1993, 'Term Rewriting in CT_Σ'. In: M. C. Gaudel and J. P. Jouannaud (eds.): *Proceedings of TAPSOFT'93, Theory and Practice of Software Development / 4th International Joint Conference CAAP/FASE*. Berlin, Heidelberg: Springer, pp. 468–484.

Corradini, A. and F. Drewes: 1997, '(Cyclic) Term Graph Rewriting is Adequate for Rational Parallel Term Rewriting'. Technical Report TR-97-14, Dipartimento di Informatica, Pisa.

Corradini, A. and F. Gadducci: 1999, 'Rewriting on Cyclic Structures: Equivalence of Operational and Categorical Descriptions'. *Theoretical Informatics and Applications* **33**, 467–493.

Jouannaud, J.-P. and H. Kirchner: 1984, 'Completion of a set of rules modulo a set of equations'. In: *Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. pp. 83–92, ACM Press.

Kennaway, J. R., J. W. Klop, M. R. Sleep, and F. J. de Vries: 1991, 'Transfinite Reductions in Orthogonal Term Rewriting Systems'. In: *Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy)*. also Report CS-R9041, CWI, 1990.

Kennaway, J. R., J. W. Klop, M. R. Sleep, and F. J. de Vries: 1994, 'On the Adequacy of Graph Rewriting for Simulating Term Rewriting'. *ACM Transactions on Programming Languages and Systems* **16**(3), 493–523.

Kennaway, J. R., J. W. Klop, M. R. Sleep, and F. J. de Vries: 1997, 'Infinitary Lambda Calculus'. *Theoretical Computer Science* **175**(1), 93–125.

Kirchner, C. (ed.): 1990, *Unification*. Academic Press inc.

Klop, J. W.: 1980, 'Combinatory Reduction Systems'. Ph.D. thesis, CWI.

Ohlebusch, E.: 1998, 'Church-Rosser Theorems for Abstract Reduction Modulo an Equivalence Relation.'. In: T. Nipkow (ed.): *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA-98)*, Vol. 1379 of *Lecture Notes in Computer Science*. pp. 17–31, Springer.

Parigot, M.: 1992, '$\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction'. In: A. Voronkov (ed.): *Proceedings of LPAR'92, Logic Programming and Automated Reasoning, St Petersburg, Russia, July 1992*, Vol. 624 of *Lecture Notes in Artificial Intelligence*. Berlin: Springer-Verlag, pp. 190–201.

Peterson, G. and M. E. Stickel: 1981, 'Complete Sets of Reductions for Some Equational Theories'. *Journal of the ACM* **28**, 233–264.

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

Sleep, M. R., M. J. Plasmeijer, and M. C. J. D. van Eekelen (eds.): 1993, *Term graph rewriting: theory and practice*. London: Wiley.

Turner, D. A.: 1979, 'A new implementation technique for applicative languages'. *Software and Practice and Experience* **9**, 31–49.

Wack, B.: 2003, 'Klop counter example in the Rho-Calculus'. Draft notes, LORIA, Nancy.