ANDREA CORRADINI

FABIO GADDUCCI

## Rewriting on cyclic structures : equivalence between the operational and the categorical description

# REWRITING ON CYCLIC STRUCTURES: EQUIVALENCE BETWEEN THE OPERATIONAL AND THE CATEGORICAL DESCRIPTION *

ANDREA CORRADINI[1] AND FABIO GADDUCCI[2]

**Abstract.** We present a categorical formulation of the rewriting of possibly cyclic term graphs, based on a variation of algebraic 2-theories. We show that this presentation is equivalent to the well-accepted operational definition proposed by Barendregt *et al.* – but for the case of *circular redexes*, for which we propose (and justify formally) a different treatment. The categorical framework allows us to model in a concise way also automatic garbage collection and rules for sharing/unsharing and folding/unfolding of structures, and to relate term graph rewriting to other rewriting formalisms.

**Résumé**. Nous présentons une formulation catégorique de la réécriture des graphes cycliques des termes, basée sur une variante de 2-théorie algébrique. Nous prouvons que cette présentation est équivalente à la définition opérationnelle proposée par Barendregt *et al.*, mais pas dans le cas des radicaux circulaires, pour lesquels nous proposons (et justifions formellement) un traitement différent. Le cadre catégoriel nous permet de modeler également la "garbage collection" automatique, et des règles de "sharing/unsharing" et "folding/unfolding" des structures. En outre, ce cadre nous permet d'exploiter pour associer la réécriture des graphes des termes à d'autres formalismes de réécriture.

## INTRODUCTION

The classical theory of *term graph rewriting* studies the issue of representing terms as directed graphs, and of modelling *term rewriting via* graph rewriting (we refer for a survey to the book [53] and the references therein). With respect to the standard representation of terms as trees, the main operational appeal of using graphs is that the sharing of common sub-terms can be represented explicitly. Intuitively, the rewrite process is speeded up, because rewriting steps do not have to be repeated for each copy of an identical sub-term. Also, by allowing term graphs with cycles, one can represent in a finitary way certain structures that arise when dealing with recursive definitions (as for the implementation of the fixed point combinator Y proposed in [58]). For these reasons term graph rewriting is often used, in the implementation of functional languages [48].

However, in our opinion there is an unsatisfactory gap between the achievements of the theories of term and of term graph rewriting. In particular, a solid ground for the theory of term rewriting is provided by the existence of three different yet equivalent characterisations, namely the *operational*, classical one (described in terms of redexes and substitutions [42]), the *logical* one (we think of the *rewriting logic* formalism [44]), and the *categorical* one, based on *algebraic (cartesian) 2-theories* [49]. While the operational description is the most suited for implementation purposes, both the logical and the categorical ones provide an *inductive* definition of the rewrite relation over terms, that lays the ground for the development of proof and analysis techniques based on structural induction. Moreover, the categorical account has the advantage of being independent from representation details, stressing the intrinsic algebraic structure of terms and their rewriting; in fact, one can safely claim that the essential structure of the collection of terms over a given signature is "cartesianity". It is worth noting that a similar "triangular" correspondence also exists for other rule based formalisms like for example *Petri nets*, where the operational definition of, say [50] parallels a categorical counterpart based on *symmetric strict monoidal categories* [45] and a logical one based on (the multiplicative fragment of) linear logic (see [43] for a survey).

Much less satisfactory are the achievements of the theory of term graph rewriting, which has been studied by many authors, but only in operational style. In fact, term graphs have been represented as directed graphs satisfying a number of constraints [3], as suitable labelled hyper-graphs called *jungles* [36], or as sets of recursive equations [1], among others. Only recently we have shown in [13] that the rewriting of *acyclic* term graphs can also be presented in a categorical way similar to the 2-categorical presentation of term rewriting. In fact (ranked, acyclic) term graphs over a signature $\Sigma$ are in one-to-one correspondence with the arrows of the free *gs-monoidal* category generated by $\Sigma$ (this result is presented in [15], where we also explain the origin of the acronym *gs-*, which stays for *graph substitution*). And acyclic term graph rewriting sequences over a rewriting system $\mathcal{R}$ (according to the definition in [3], the most widely accepted in the literature) are faithfully represented by the cells of the free gs-monoidal 2-category generated by a suitable representation of the rules of $\mathcal{R}$.

The gs-monoidal categories are symmetric strict monoidal categories equipped with two transformations, the *duplicator* "$\nabla$" (read *nabla*) and the *discharger* "!" (*bang*), from which cartesian categories can be recovered requiring their naturality. As shown in [15], the non-naturality of $\nabla$ is related to the fact that term graphs with different degree of sharing are distinct, while that of ! allows for the presence of garbage in a term graph (*i.e.*, nodes not reachable from the roots).

The main contribution of this paper is the generalisation of [13] to the categorical representation of *possibly cyclic* term graph rewriting. This is not a minor point, since it is shown in [30] that in the presence of suitable, quite natural, sharing strategies, cyclic term graphs can be generated during rewriting even if starting from an acyclic graph and rules are acyclic. In this paper we stick to acyclic rules, yet allowing for arbitrary cycles in the term graphs which are rewritten.

In Section 1 we first introduce *(possibly cyclic) ranked term graphs*, and three operations on them: *composition* (a counterpart of term substitution), *(disjoint) union* and *feedback*, then showing that every term graph can be obtained as the value of an expression containing "atomic" term graphs as constants, and composition, union and feedback as operators. With respect to a similar result for acyclic term graphs in [15], two are the main differences. Firstly, the feedback operation is new, and it is exactly what we need to generate cycles. Secondly, a new kind of nodes (called "$\perp$-nodes") shows up, which were not needed in the treatment of acyclic term graphs. Such nodes are produced by the reduction of "circular redexes", and they can safely be considered as "cycles of length zero".

In Section 2 we present our definition of term graph rewriting, a little variation of that in [3] (their relationship is analysed in Sect. 5). In Section 3 we recall the basic definitions about *gs-monoidal 2-categories*, introducing their *traced* counterpart: to this aim, we extend to the 2-categorical level the presentation of the "feedback" operator in [21], investigated in more generality in [37]. The traced structure is the categorical counterpart of the feedback operation on term graphs, and cyclic term graphs over $\Sigma$ can be represented as the arrows of the free traced gs-monoidal category generated by $\Sigma$. Extending the corresponding result for acyclic graphs in [13], in Section 4 we prove that if the term graph rules of a rewriting system are represented as cells over this category, and a traced gs-monoidal 2-category is freely generated by them, the resulting cells faithfully represent the term graph rewriting sequences of the system satisfying a mild restriction.

An interesting consequence of this result is that by representing (cyclic) term graphs as arrows of a suitable free category, we highlight the intrinsic algebraic structure of such term graphs, independently of a specific representation: like "cartesianity" is the essential structure of terms, so "traced gs-monoidality" turns out to be the essential structure of (cyclic) term graphs. Even if 2-categories equivalent to ours have been considered independently in [34, 46], in both works the focus is the relation between such categorical structures and a *logical* presentation of term graph rewriting. On the contrary, to our knowledge our correspondence result is the first that explicitly and formally relates a categorical and an operational description of term graph rewriting, thus completing a triangular correspondence analogous to those for term rewriting and Petri nets mentioned above.

In Section 5 we focus on the differences between our definition of term graph rewriting and that originally proposed in [3], related to the different handling of *circular redexes* and to *garbage collection*. In particular, in Section 5.1 we explain the disagreement with [3] on circular redexes. The paradigmatic example of such redexes is the application of rule $R_I : I(x) \rightarrow x$ to the graph having one node labelled $I$ and one looping edge (the $I$-loop): the point is that according to the definition of rewriting in [3], the $I$-loop reduces to itself, while using our definition it reduces to a $\perp$-node, and this fact is explained in terms of the categorical structure. In Section 5.2 we show that small modifications of the formal definitions allow us to capture term graph rewriting with automatic garbage collection as well. In Section 5.3 we discuss how to introduce rules for the automatic folding or unfolding of structures, showing that this allows us to relax a technical restriction on redexes in the main correspondence result. In Section 5.4 we summarise the relationship between traced gs-monoidal 2-categories and other 2-categorical models of term (graph) rewriting proposed in the literature using suitable adjunctions, relating *e.g.* term graph rewriting to rational term rewriting (as introduced in [14]) by exploiting the categorical framework. Lastly, in Section 5.5 we sketch an historical overview of the various algebraic characterisation of the notion of fixed point, mostly related to *iteration theories* [7].

# 1. (Cyclic) term graphs

This section introduces (ranked, possibly cyclic) term graphs as isomorphism classes of (ranked) labelled graphs. Since our main concern is to stress the underlying algebraic structure, the following presentation of term graphs slightly departs from the standard definition of [3], as explained below.

**Definition 1.1** (graphs). Let $\Sigma$ be a (one-sorted) signature, *i.e.*, a ranked set of operator symbols, and let *arity* be the function returning the arity of an operator symbol, *i.e.*, arity$(f) = n$ iff $f \in \Sigma_n$. A *labelled graph d (over $\Sigma$)* is a triple $d = \langle N, l, s \rangle$, where $N$ is a set of *nodes*, $l : N \rightharpoonup \Sigma$ is a partial function called the *labelling function*, $s : N \rightharpoonup N^*$ is a partial function called the *successor function*, and such that the following conditions are satisfied:

- $dom(l) = dom(s)$, *i.e.*, labelling and successor functions are defined on the same subset of $N$; a node $n \in N$ is called *empty* if $n \notin dom(l)$;
- for each node $n \in dom(l)$, $arity(l(n)) = length(s(n))$, *i.e.*, each non-empty node has as many successor nodes as the arity of its label.

If $s(n) = \langle n_1, \ldots, n_k \rangle$, we say that $n_i$ is the $i$-th *successor* of $n$ and denote it by $s(n)_i$. A labelled graph is *discrete* if all its nodes are empty. A *path* in $d$ is a sequence $\langle n_0, i_0, n_1, \ldots, i_{m-1}, n_m \rangle$, where $m \geq 0$, $n_0, \ldots, n_m \in N$, $i_0, \ldots, i_{m-1} \in \mathbb{N}$ (the natural numbers), and $n_k$ is the $i_{k-1}$-th successor of $n_{k-1}$ for $k \in \{1, \ldots, m\}$. The *length* of this path is $m$; if $m > 0$, the path is *proper*. A *cycle* is a proper path like above where $n_0 = n_m$. If $a \in N$ is a node of a graph $d = \langle N, l, s \rangle$, then by $d|a$ we denote the full subgraph of $d$ containing all the nodes reachable from $a$ *via* a path.

For a graph $d$ we often denote its components by $N(d)$, $l_d$ and $s_d$, respectively. Moreover, $N_\emptyset(d)$ and $N_\Sigma(d)$ denote the set of empty and non-empty nodes of $d$, respectively (thus $N(d) = N_\Sigma(d) \uplus N_\emptyset(d)$, for $\uplus$ disjoint union).

**Definition 1.2** (graph morphisms, category $\mathbf{G}_\Sigma$). Let $d$ and $d'$ be two graphs. A *(graph) morphism* $f : d \to d'$ is a function $f : N(d) \to N(d')$ that preserves labelling and successors, *i.e.*, such that for each node $a \in N_\Sigma(d)$, $l_{d'}(f(a)) = l_d(a)$, and $s_{d'}(f(a))_i = f(s_d(a)_i)$ for each $i \in \{1, \dots, arity(l_d(a))\}$. A morphism $f : d \to d'$ is *$\Sigma$-injective* if its restriction to $N_\Sigma(d)$ is injective.

Graphs over $\Sigma$ and graph morphisms form a category denoted $\mathbf{G}_\Sigma$.

For each $i \in \mathbb{N}$, we shall denote by $\underline{i}$ the set $\underline{i} = \{1, \dots, i\}$ (thus $\underline{0} = \emptyset$).

**Definition 1.3** (ranked graphs and term graphs). An *$(i, j)$-ranked graph* (or also, a *graph of rank $(i, j)$*) is a triple $g = \langle r, d, v \rangle$, where $d$ is a graph with at least $j$ empty nodes, $r : \underline{i} \to N(d)$ is a function called the *root mapping*, and $v : \underline{j} \to N_\emptyset(d)$ is an injective function from $\underline{j}$ to the empty nodes of $d$, called the *variable mapping*. Node $r(k)$ is called the *k-th root* of $d$, and $v(k)$ is called the *k-th variable* of $d$, for each admissible $k$. An empty node which is not in the image of the variable mapping is called a *$\perp$-node*, for reasons which will be discussed briefly in Section 5.4.

Two $(i, j)$-ranked graphs $g = \langle r, d, v \rangle$ and $g' = \langle r', d', v' \rangle$ are *isomorphic* if there exists a *ranked graph isomorphism* $\phi : g \to g'$, *i.e.*, a graph isomorphism $\phi : d \to d'$ such that $\phi \circ r = r'$ and $\phi \circ v = v'$.

An *$(i, j)$-ranked term graph $G$* (or *with rank $(i, j)$*) is an isomorphism class of $(i, j)$-ranked graphs. We shall often write $G_j^i$ to recall that $G$ has rank $(i, j)$.

Term graphs are defined in the seminal paper [3] as labelled graphs (as in Def. 1.1) with a distinguished node, the *root*. Thus there are two differences with the above definition. Firstly, our notion of term graph looks "more abstract", because it is defined as an isomorphism class of graphs. This allows us to disregard the concrete identity of nodes when manipulating term graphs, provided that the operations we introduce are well defined on isomorphism classes. Admittedly, in most of the literature about term graph rewriting, isomorphic graphs are informally considered as equivalent (for example, the identity of nodes is not depicted in the usual graphical representation of term graphs), thus our definition just formalises a common way of "reasoning up-to isomorphism".

Secondly, our term graphs are ranked: we equip term graphs with lists of distinguished nodes in order to define composition operations on them. In [3] this technique is not used simply because it is not needed for the goals of the paper, and the single root used there has a different role, being used to relate a term graph with a term unfolded from it. Summarising, we can safely see our term graphs as a minor variation of those in [3].

We introduce now three operations on ranked term graphs. The *composition* of two ranked term graphs is obtained by glueing the variables of the first one with the roots of the second one, and it is defined only if their number is equal. This operation allows us to define a category having ranked term graphs as arrows.

Next, the *union* of term graphs is introduced: it is always defined, and it is a sort of disjoint union where roots and variables are suitably renumbered. Finally, the *feedback* over a term graph with at least one variable and one root is defined as the introduction of a connection from the rightmost variable to the rightmost root, possibly resulting in a cycle. These three operations provide ranked term graphs with a traced monoidal structure, that will be made explicit in Section 4.

**Definition 1.4** (operations on ranked term graphs). **Composition.** The *composition* of two term graphs $G'^j_k = [\langle r', d', v' \rangle]$ and $G^i_j = [\langle r, d, v \rangle]$ is the term graph $G'^j_k; G^i_j$ of rank $(i, k)$ obtained by glueing the variables of $G$ with the corresponding roots of $G'$. Formally, $G'^j_k; G^i_j$ is defined as $[\langle in_d \circ r, d'', in_{d'} \circ v' \rangle]$, where $\langle d'', in_d : d \to d'', in_{d'} : d' \to d'' \rangle$ is the (chosen) pushout of $\langle v : \underline{j} \to d, r' : \underline{j} \to d' \rangle$ in category $\mathbf{G}_\Sigma$ (set $\underline{j}$ is regarded as a discrete graph)[1].

**Union.** The *union* or *parallel composition* of two term graphs $G^i_j = [\langle r, d, v \rangle]$ and $G'^k_l = [\langle r', d', v' \rangle]$ is the term graph $G^i_j \oplus G'^k_l$ of rank $(i + k, j + l)$ obtained by the disjoint union of the components of $G^i_j$ and $G'^k_l$. Formally, $G^i_j \oplus G'^k_l$ is defined as $[\langle r \oplus r', d \uplus d', v \oplus v' \rangle]$, where $d \uplus d'$ is the (chosen) coproduct of $d, d'$ in category $\mathbf{G}_\Sigma$, $r \oplus r' : \underline{i + k} \to d \uplus d'$ is the unique morphism induced by $r$ and $r'$, and $v \oplus v' : \underline{j + l} \to d \uplus d'$ is defined similarly.

**Feedback.** The *feedback* over a term graph $G^{i+1}_{j+1} = [\langle r, d, v \rangle]$ is the term graph $(G^{i+1}_{j+1})^\dagger$ of rank $(i, j)$ obtained by glueing the $j + 1$-th variable of $G$ with its $i + 1$-th root. Formally, $(G^{i+1}_{j+1})^\dagger$ is defined as $[\langle u_d \circ r \circ in_r, d', u_d \circ v \circ in_v \rangle]$, where $\langle d', u_d \rangle$ is the (chosen) coequalizer of $r_f, v_f : \underline{1} \to d$ (with $r_f(1) = r(i + 1)$ and $v_f(1) = v(j + 1)$), $in_r : \underline{i} \to \underline{i + 1}$ is the inclusion morphism such that $in_r(x) = x$, and $in_v : \underline{j} \to \underline{j + 1}$ is defined similarly.

In the rest of the paper we adopt the following notation for repeated applications of feedback: $G^{\dagger^0}$ denotes $G$, and $G^{\dagger^{n+1}}$ denotes $(G^{\dagger^n})^\dagger$. Note also that the various operators are defined "explicitly" on the concrete graphs contained in the equivalence classes, as indicated by our use of *chosen* colimits. Thus, given $g \in G$, we will sometimes denote $g^\dagger$ for the representative of the graph in $G^\dagger$ uniquely obtained from $g$ (see *e.g.* Lem. 4.7), and similarly for the other operators.

**Example 1.5** (term graphs, composition, union and feedback). In    the    term graphs shown in Figure 1, variable nodes are represented by the natural numbers corresponding to their position in the list of variables, and are depicted as a vertical sequence on the left; $\perp$-nodes are drawn as $\perp$, as expected; non-empty nodes are represented by their label, from where the edges pointing to the successors leave; the list of numbers on the right represent pointers to the roots: a dashed arrow from $j$ to a node indicates that it is the $j$-th root.

---

[1] Actually, the pushout of two arrows in $\mathbf{G}_\Sigma$ (as well as the coequalizer, see later) does not always exist: it does however in the case we are interested in, since the morphism $v : \underline{j} \to d$ is injective and has only empty nodes in the codomain. See [17] for necessary and sufficient conditions for their existence in the equivalent category of *jungles*.
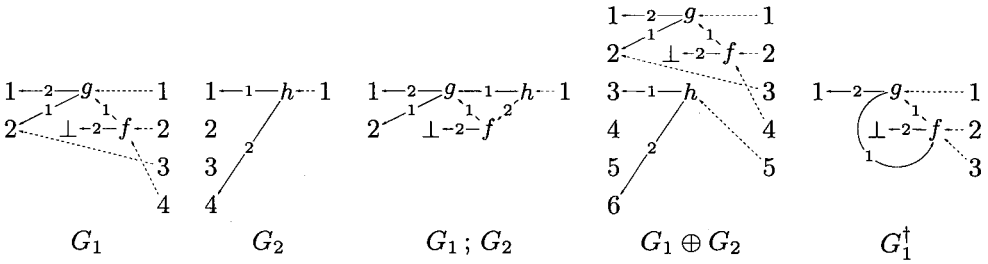
FIGURE 1. Two term graphs and some operations on them.

For example, the first term graph $G_1$ has rank $(4,2)$, five nodes (three empty $(1, 2, \text{ and } \perp)$ and two non-empty, $f$ and $g$), the successors of $g$ are the variables 2 and 1 (in this order), the successors of $f$ are $g$ and $\perp$, and the four roots are $g$, $f$, 2, and $f$. The second term graph $G_2$ has rank $(1,4)$, five nodes (four empty $(1, 2, 3, \text{ and } 4)$ and one non-empty, $h$), the successors of $h$ are the variables 1 and 4 (in this order), and the only root is $h$.

Composition of term graphs can be performed by matching the roots of the first graph with the variables of the second one, and then by eliminating them, as shown by $G_1 ; G_2$. The last two term graphs show the result of union and feedback. Intuitively, the feedback over a term graph $G$ is obtained by "redirecting" the pointers from the last variable of $G$ to its last root, and then by eliminating both the last variable and the last root (as it can be grasped by comparing $G_1$ and $G_1^\dagger$). If the last variable coincides with the last root, it follows from the formal definition that the corresponding node becomes a $\perp$-node, since it remains an empty node, but it lies now outside of the range of the variable mapping.

In [15] we presented a decomposition result for ranked, *acyclic* term graphs containing no $\perp$-nodes, obtained as the value of an expression containing suitable *atomic* term graphs as constants, and composition and union as operators. We extend this result to the possibly cyclic case, showing that an arbitrary term graph can be obtained by using also the operation of feedback.

**Definition 1.6** (atomic term graphs). An *atomic term graph* is a term graph in $\{G_f \mid f \in \Sigma\} \cup \{G_{id}, G_\rho, G_\nabla, G_!, G_\emptyset\}$, which are depicted in Figure 2.

Since every node of an atomic term graph is a root or a variable, such term graphs can be formally defined as follows (using, a bit improperly, $r, v, s, \text{ and } l$ for the root, variable, successor, and labelling functions, respectively):

- $G_f$ has rank $(1, j)$, with $l(r(1)) = f$, and $s(r(1))_x = v(x)$ for $x \in \underline{j}$, for each $f \in \Sigma$ with $arity(f) = j$.
- $G_{id}$ has rank $(1, 1)$, with $r(1) = v(1)$.
- $G_\rho$ has rank $(2, 2)$, with $r(1) = v(2)$ and $r(2) = v(1)$.
- $G_\nabla$ has rank $(2, 1)$, with $r(1) = r(2) = v(1)$.
- $G_!$ has rank $(0, 1)$, one empty node, and no roots.
- $G_\emptyset$ is the empty graph having rank $(0, 0)$.

$G_f$ : 
$1 \overset{1}{\underset{2}{\longrightarrow}} f \cdots 1$
$2 \nearrow$
$\vdots$
$j$

$G_{id}$ : $1 \cdots\cdots 1$

$G_\rho$ : $1 \searchrightarrow 1$
$2 \nearrow\searrow 2$

$G_\nabla$ : $1 \cdots\cdots 1$
$\searrow 2$

$G_!$ : $1$

$G_\emptyset$ : $\emptyset$
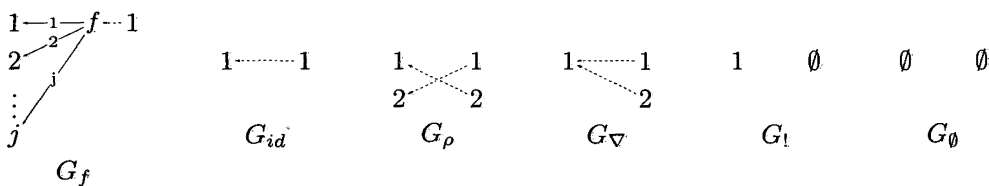
$\emptyset$ $\emptyset$

FIGURE 2. Atomic term graphs.

A *term graph expression* is a term over the signature containing all ranked term graphs as constants, union and composition as binary operators, and feedback as unary operator. An expression is *well-formed* if all occurrences of composition and feedback are defined for the rank of the argument sub-expressions, according to Definition 1.4; its *rank* is then computed inductively from the rank of the term graphs appearing in it, and its value is the term graph obtained by evaluating all operators in it.

**Lemma 1.7** (decomposition of acyclic term graphs). *Every acyclic term graph containing no $\perp$-nodes can be obtained as the value of a well-formed term graph expression containing atomic term graphs as constants and composition and union as operators.*

*Proof.* See [15], Theorem 9. □

**Proposition 1.8** (decomposition of term graphs). *Every term graph can be obtained as the value of a well-formed term graph expression containing atomic term graphs as constants, and composition, union and feedback as operators.*

*Proof.* The proof is by induction on the number of cycles. For the base case, first notice that $G_\perp \overset{def}{=} (G_\nabla)^\dagger$ is the term graph of rank $(1,0)$ consisting of a single $\perp$-node which is the only root. Now suppose that $G$ of rank $(i,j)$ has no cycle and, say, $k$ $\perp$-nodes. Let $G'$ of rank $(i,j+k)$ be obtained from $G$ by adding its $k$ $\perp$-nodes as variables $j+1,\ldots,j+k$. Then it is easy to check that $G = G_{id}^{(j)} \oplus G_\perp^{(k)} ; G'^2$. The statement then follows by Lemma 1.7, because $G'$ is acyclic and has no $\perp$-nodes.

For the inductive case, let $G$ be a term graph of rank $(i,j)$ with at least one cycle $\rho = \langle a_0, i_0, a_1, \ldots, i_{m-1}, a_m \rangle$. Let $G'$ be the term graph of rank $(i, j + k)$ obtained from $G$ by making all the non-empty nodes in the subgraph $G|a_1$ empty, and adding them as the $j + 1, \ldots, j + k$-th variables. Furthermore, let $G''$ be the term graph of rank $(j+k,j)$ which is the subgraph of $G$ containing all its variables and $G|a_1$, and where all the nodes are roots in the same order in which they appear as variables in $G'$. Then $G = G'' ; G'$. Finally, let $G_u$ be the term graph of rank $(j + k + 1, j + 1)$ obtained from $G''$ by adding a new root (the $(j + k + 1)$-th) pointing to $a_1$, a new variable (the $(j+1)$-th), and such that $s_u(a_0)_{i_0} = v_u(j+1)$, that is, the successor in $G_u$ at position $i_0$ of $a_0$ is the new variable. Then it is easily seen that $G'' = (G_u)^\dagger$, and the statement follows by induction hypothesis on $G'$ and $G_u$, since both have at least one cycle less than $G$. □

___

[2]For a term graph $H$, $H^{(0)}$ denotes $G_\emptyset$, and $H^{(n+1)} = H^{(n)} \oplus H$.

## 2. TERM GRAPH REWRITING

Following [3], a term graph rewriting rule is defined as a single (acyclic) graph with two roots, which is intended to represent not only the left- and right-hand sides (corresponding to the subgraph rooted at the first and second root, respectively), but also the fact that the two sides can share some subgraph.

**Definition 2.1** (rule, redex, rewriting system). A *rule* (over $\Sigma$) is a triple $\mathbf{R} = \langle d, r_l, r_r \rangle$, where $d$ is an acyclic graph (over $\Sigma$) having no $\perp$-nodes, and $r_l, r_r$ are two distinct nodes of $d$, called the *left root* and the *right root*, respectively. Furthermore, we require that (1) node $r_l$ is not empty, (2) all nodes of $d$ are reachable from $r_l$ or $r_r$, and (3) all empty nodes in $d$ are reachable from $r_l$.

A *redex* in a graph $d_0$ is a pair $\Delta = \langle \mathbf{R}, f \rangle$ where $\mathbf{R} = \langle d, r_l, r_r \rangle$ is a rule, and $f : d|r_l \to d_0$ is a graph morphism which is *non-self-overlapping*, *i.e.*, such that for each $a \in N_\Sigma(d|r_l) - \{r_l\}$, $f(a) \neq f(r_l)$. Node $f(r_l)$ of $d_0$ is called the *root* of the redex $\Delta$. A redex $\langle \mathbf{R}, f \rangle$ is $\Sigma$-*injective* if so is $f$.

A *(term graph) rewriting system* is a pair $\mathcal{R} = \langle \{\mathbf{R}_i\}_{i \in I}, \Sigma \rangle$ such that $\mathbf{R}_i$ is a rule over $\Sigma$ for all $i \in I$.

Conditions (1) and (3) correspond directly to analogous, standard restrictions for term rewriting rules (recalling that empty nodes are essentially variables), while (2) is a "no-junk" condition.

A redex in a graph $d_0$ specifies where a given rule can be applied, and the application of the rule consists essentially of deleting the root of the redex, and of adding a fresh copy of the right-hand side to the resulting graph, connecting it in a suitable way. The (images of the) other nodes in the left-hand side of the rule (*i.e.*, those in $N_\Sigma(d|r_l) - \{r_l\}$) are necessary for the application of the rule, but are not deleted by the rewriting, because in general they can be shared in $d_0$: as such they can be considered as *read-only* resources or *context conditions* [47]. Under this interpretation *self-overlapping* redexes are not allowed because they provide an inconsistent specification: if $a \in N_\Sigma(d|r_l) - \{r_l\}$ and $f(a) = f(r_l)$, then the same node should be deleted and preserved at the same time[3].

We are now ready to introduce our operational definition of term graph rewriting. The relation between this definition and the corresponding one in [3] is discussed in Section 5.

**Definition 2.2** (term graph rewriting). Let $g_0 = \langle r_0, d_0, v_0 \rangle$ be an $(i, j)$-ranked graph, and let $\Delta = \langle \langle d, r_l, r_r \rangle, f : d|r_l \to d_0 \rangle$ be a redex. We say that $g_0$ *rewrites to* $g_4$ *via* $\Delta$, denoted as $g_0 \to_\Delta g_4$, if $g_4$ is the $(i, j)$-ranked graph $\langle tr_\Delta \circ r_0, d_4, tr_\Delta \circ v_0 \rangle$, whose components are obtained through the construction below.

　　[**Root undefine phase**]: Let $d_1$ be the graph obtained from $d_0$ by making the root of the redex empty: that is, $l_{d_1}$ and $s_{d_1}$ are undefined on $f(r_l)$, and agree with $l_{d_0}$ and $s_{d_0}$ elsewhere.

---

[3]The restriction to non-self-overlapping redexes corresponds to the *identification condition* in the double-pushout approach to graph rewriting [16]. Even if Definition 2.2 could be applied as it is to self-overlapping redexes, the main result of the paper would not hold anymore.

[**Build phase**]: Let $d_2$ be the graph obtained by adding to $d_1$ (a copy of) the part of the rule graph $d$ not reachable from $r_l$: that is, $N(d_2) = N(d_1) \uplus (N(d) - N(d|r_l))$, and $l_{d_2}$ and $s_{d_2}$ are determined in the expected way from $l_{d_1}$, $l_d$, and $s_{d_1}$, $s_d$, respectively. And let $\hat{f} : N(d) \to N(d_2)$ be the obvious extension of function $f : N(d|r_l) \to N(d_0)$.

[**Redirection phase**]: Let $d_3$ be the graph obtained by replacing in $d_2$ all references to the root of the redex by references to the image of the right root: that is, $d_3 = \langle N(d_2), l_{d_2}, s_{d_3}\rangle$, where $s_{d_3}(a)_i = s_{d_2}(a)_i$ if $s_{d_2}(a)_i \neq f(r_l)$, and $s_{d_3}(a)_i = \hat{f}(r_r)$ if $s_{d_2}(a)_i = f(r_l)$.

[**Root removal phase**]: Let $d_4$ be the graph obtained from $d_3$ by removing node $f(r_l)$—unless $f(r_l) = \hat{f}(r_r)$, in which case $d_4 = d_3$. And let the *track function* $tr_\Delta : N(d_0) \to N(d_4)$ be defined as $tr_\Delta(a) = \hat{f}(r_r)$ if $a = f(r_l)$, and $tr_\Delta(a) = a$ otherwise.

We will say that $\Delta = \langle \mathbf{R}, f\rangle$ is a redex in a term graph $G$ if it is a redex in one of its elements, say $g = \langle r, d, v\rangle$. In this case, if $g \to_\Delta g'$ then we write $G \to_\Delta G'$, where $G' = [g']$, or simply $G \to_\mathbf{R} G'$ if $f$ is clear from the context[4].

A *term graph rewriting sequence* $G_0 \to_\mathcal{R}^* G_n$ over a rewriting system $\mathcal{R}$ is a sequence of $n \geq 0$ rewriting steps $G_0 \to_{\Delta_1} G_1 \cdots G_{n-1} \to_{\Delta_n} G_n$, where at each step a rule of $\mathcal{R}$ is used. A rewriting sequence is $\Sigma$-*injective* if all redexes in it are.

**Example 2.3** (rewriting sequence). Let $\Sigma$ be the signature containing the constants $\{a, b, \emptyset\}$, the unary operators $\{@, n, l, r\}$ and the binary operator $p$. Term graphs over $\Sigma$ are intended to represent manipulations on lists of $a$'s and $b$'s, where $\emptyset$ is the empty list, $p(air)$ is the pairing operator, $l(eft)$ and $r(ight)$ are the obvious selectors, $@$ is a pointer and $n(ext)$ moves a pointer. The top row of Figure 3 shows the term graphs representing rules $\mathbf{R_1} : n(@(p(x, y))) \to @(y)$ and $\mathbf{R_2} : r(p(x, y)) \to y$; rule $\mathbf{R_2}$ is called "collapsing" because its right root is a variable. The rest of the figure shows the rewriting sequence $G_0 \to_{\mathbf{R_1}} G_3 \to_{\mathbf{R_2}} G_4$, where also some intermediate states are depicted. A node enclosed in a square is the left root (in the rules) or the root of the redex (in the other graphs), while a node enclosed in a circle is the right root (in the rules) or its image. Term graphs $G_1$ and $G_2$ are the results of the *root undefine* and *build* phase, respectively, of the rewriting $G_0 \to_{\mathbf{R_1}} G_3$. Note furthermore that $G_4$ is obtained from $G_3$ *via root undefine*, and the remaining phases of the rewriting $G_3 \to_{\mathbf{R_2}} G_4$ leave it unchanged because the images of the left and right root coincide.

## 3. TRACED GS-MONOIDAL 2-CATEGORIES

We open this section recalling some basic definitions about 2-categories [38,39]. We then introduce *gs-monoidal 2-categories* [13], and their *traced* extension, that will be used in our characterisation of term graph rewriting.

---

[4]The extension is sound, since for each $h \in G$ and ranked graph isomorphism $\phi : g \to h$, $\Delta$ induces a redex in $h$, obtained by composing $f$ with $\phi$. Furthermore, if $g \to_\Delta g'$ and $h \to_{\langle \mathbf{R}, \phi \circ f\rangle} h'$, then it is easy to check that $g'$ and $h'$ are isomorphic.
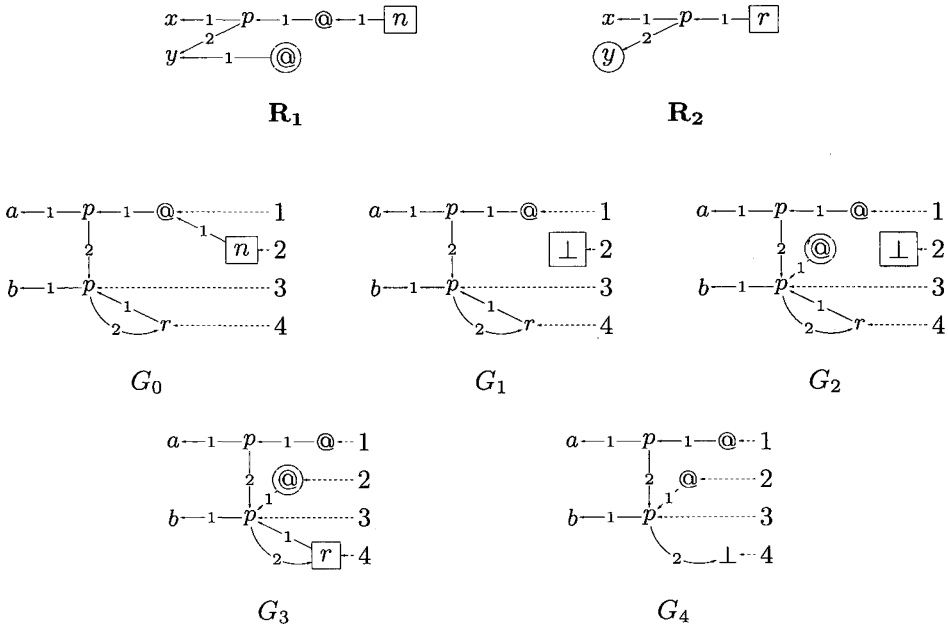
FIGURE 3. A sample term graph rewriting sequence.

**Definition 3.1** (2-categories). A *2-category* $\underline{\mathbf{C}}$ is a structure $\langle Ob_c, \mathbf{C}, *, id \rangle$ such that $Ob_c$ is a set of *2-objects* and, indexed by elements in $Ob_c$, $\mathbf{C}$ is a family of categories $\mathbf{C}[a, b]$ (the *hom-categories* of $\underline{\mathbf{C}}$), $*$ is a family of functors $*_{a,c}^b$ : $\mathbf{C}[a, b] \times \mathbf{C}[b, c] \to \mathbf{C}[a, c]$ and $id$ is a family of objects $id_a \in |\mathbf{C}[a, a]|$, satisfying for each $\alpha \in \mathbf{C}[a, b]$, $\beta \in \mathbf{C}[b, c]$, and $\gamma \in \mathbf{C}[c, d]$

[**category**]      $id_a * \alpha = \alpha = \alpha * id_b$      $(\alpha * \beta) * \gamma = \alpha * (\beta * \gamma)$

where for the sake of readability the indexes of $*$ are dropped, and the arrow $id_{id_a}$ corresponding to the identity on the object $id_a$ is denoted by the object itself.

The *underlying category* of $\underline{\mathbf{C}}$, denoted $\mathbf{C}_u$, has elements of $Ob_c$ as objects, and objects of the hom-category $\mathbf{C}[a, b]$ as elements of the hom-set $\mathbf{C}_u[a, b]$: $\mathbf{C}_u$ is well-defined thanks to the *category* axioms. As usual, we call *arrows* and *cells* of the 2-category $\underline{\mathbf{C}}$ the objects and arrows of the hom-categories, respectively. By $\alpha : f \Rightarrow g : a \to b$ we mean that $\alpha$ is a cell in $\mathbf{C}[a, b]$ from $f$ to $g$.

For the sake of brevity, we present the following definitions directly at the 2-categorical level: the analogous definitions for (ordinary) categories are easily obtained as a special case.

**Definition 3.2** (2-functors and (natural) 2-transformations). Let $\underline{\mathbf{C}}$, $\underline{\mathbf{D}}$ be 2-categories. A *2-functor* $F : \underline{\mathbf{C}} \to \underline{\mathbf{D}}$ is a pair $\langle F_o, F_m \rangle$, where $F_o : Ob_c \to Ob_d$ is a function and $F_m$ is a family of functors $F_{a,b} : \mathbf{C}[a, b] \to \mathbf{C}[F_o(a), F_o(b)]$ commuting in the expected way with $*$ and preserving $id$.

Let $F, G : \underline{C} \to \underline{D}$ be 2-functors. A *2-transformation* $\eta : F \Rightarrow G$ is a family of arrows of $\underline{D}$, indexed by objects of $\underline{C}$, such that $\eta_a \in |\mathbf{D}[F_o(a), G_o(a)]|$; it is *natural* if moreover it verifies $F_{a,b}(\alpha) * \eta_b = \eta_a * G_{a,b}(\alpha)$ for every cell $\alpha \in \mathbf{C}[a, b]$.

**Definition 3.3** (traced gs-monoidal 2-categories). A *gs-monoidal 2-category* is a structure $\mathcal{C} = \langle \underline{C}, \otimes, e, \rho, \nabla, ! \rangle$, where $\underline{C}$ is a 2-category and

- $e$ is a distinguished object in $\underline{C}$ and $\otimes : \underline{C} \times \underline{C} \to \underline{C}$ is a 2-functor, satisfying
  [**monoid**]        $id_e \otimes \alpha = \alpha = \alpha \otimes id_e$        $(\alpha \otimes \beta) \otimes \gamma = \alpha \otimes (\beta \otimes \gamma)$
- $\rho : \otimes \Rightarrow \otimes \circ X : \underline{C} \times \underline{C} \to \underline{C}$ is a natural 2-transformation (where X is the 2-functor that swaps its two arguments), such that $\rho_{e,e} = id_e$ and satisfying
  [**symmetry**]      $(id_a \otimes \rho_{b,c}) * (\rho_{a,c} \otimes id_b) = \rho_{a \otimes b, c}$        $\rho_{a,b} * \rho_{b,a} = id_{a \otimes b}$
- $\nabla : Id_{\underline{C}} \Rightarrow \otimes \circ \Delta : \underline{C} \to \underline{C}$ and $! : Id_{\underline{C}} \Rightarrow e : \underline{C} \to \underline{C}$ are 2-transformations ($\Delta$ is the diagonal 2-functor), such that $!_e = \nabla_e = id_e$ and satisfying
  [**duplication**]      $\nabla_a * (id_a \otimes \nabla_a) = \nabla_a * (\nabla_a \otimes id_a)$
  $\nabla_a * (id_a \otimes !_a) = id_a$        $\nabla_a * \rho_{a,a} = \nabla_a$
  [**monoidality**]    $\nabla_{a \otimes b} * (id_a \otimes \rho_{b,a} \otimes id_b) = \nabla_a \otimes \nabla_b$        $!_{a \otimes b} = !_a \otimes !_b$.

A *traced gs-monoidal 2-category* is a structure $\mathcal{C}_t = \langle \underline{C}, \otimes, e, \rho, \nabla, !, tr \rangle$, where $\mathcal{C} = \langle \underline{C}, \otimes, e, \rho, \nabla, ! \rangle$ is a gs-monoidal 2-category and

- $tr^u_{a,b} : \mathbf{C}[a \otimes u, b \otimes u] \to \mathbf{C}[a, b]$ is a family of functors indexed by 2-objects in $\underline{C}$, satisfying
  [**naturality**]      $tr^u_{a,b}(\alpha * (id_b \otimes \beta)) = tr^v_{a,b}((id_a \otimes \beta) * \alpha)$
  $tr^u_{c,d}((\alpha \otimes id_u) * \beta * (\gamma \otimes id_u)) = \alpha * tr^u_{a,b}(\beta) * \gamma$
  [**vanishing**]      $tr^e_{a,b}(\alpha) = \alpha$        $tr^v_{a,b}(tr^u_{a \otimes v, b \otimes v}(\alpha)) = tr^{v \otimes u}_{a,b}(\alpha)$
  [**superposing**]  $tr^u_{c \otimes a, c \otimes b}(id_c \otimes \alpha) = id_c \otimes tr^u_{a,b}(\alpha)$
  [**yanking**]      $tr^u_{u,u}(\rho_{u,u}) = id_u$.

A *gs-monoidal 2-functor* $F : \mathcal{C} \to \mathcal{D}$ is a 2-functor which preserves all the relevant structure "on the nose", *i.e.*, such that $F(e) = e'$, $F(\alpha \otimes \beta) = F(\alpha) \otimes' F(\beta)$, and so on. A *traced* gs-monoidal 2-functor $F : \mathcal{C}_t \to \mathcal{D}_t$ is a gs-monoidal 2-functor which preserves also the traced structure.

We denote as **TGS-2Cat** (**GS-2Cat**) the category of small traced gs-monoidal (gs-monoidal, respectively) 2-categories and corresponding 2-functors, and as **TGS-Cat** (**GS-Cat**) its full sub-category including only small categories.

Many structures similar to the 2-categories defined above have been studied in literature: we refer the reader to the concluding remarks of [13] and [15], as well as to Section 5.5 below, for a comparison between a few approaches. We sketch now how to generate from a given signature a free traced gs-monoidal category, called its *traced gs-monoidal theory*.

**Definition 3.4** (generalised signatures). A *generalised signature* $\Sigma$ is a four-tuple $\langle M, T, s, t \rangle$, where $M$ is a monoid, $T$ a set of *operators*, and $s, t : T \to U(M)$ are (the source and target) functions, for $U(M)$ the underlying set of $M$. A *morphism of generalised signatures* $f : \Sigma \to \Sigma'$ is a pair $\langle f_M : M \to M', f_T : T \to T' \rangle$, where $f_M$ is a monoid homomorphism and $f_T$ a function preserving sources and targets. Generalised signatures and their morphisms form a category, denoted **GenSig**.

Note that a one-sorted signature is a generalised signature where $M$ is the free monoid generated by a singleton (whose elements we denote by natural numbers), and $f : n \to 1$ if $arity(f) = n$.

**Proposition 3.5** (free traced gs-monoidal categories). *Let $V_t$ : **TGS-Cat** $\to$ **GS-Cat** be the functor mapping a traced gs-monoidal category to the underlying gs-monoidal one (forgetting the traced structure), and let $V_{gs}$ : **GS-Cat** $\to$ **GenSig** be the functor mapping a gs-monoidal category to the underlying generalised signature (forgetting all the structure but the monoidal structure on objects). Both functors admit a left (free) adjoint, denoted $GS$ : **GenSig** $\twoheadrightarrow$ **GS-Cat** and $T$: **GS-Cat** $\to$ **TGS-Cat**, respectively.*

*Proof.* See [15], Example 14 and Definition 18 for an explicit description of the left adjoint $GS$. Let $\mathcal{C} = \langle \mathbf{C}, \otimes, e, \rho, \nabla, ! \rangle$ be a gs-monoidal category. Then the associated traced category is the structure $T(\mathcal{C}) = \langle T(\mathbf{C}), \oplus, e, \rho, \nabla, !, tr \rangle$, where the objects of $T(\mathbf{C})$ are the same as the objects of $\mathbf{C}$, while the arrows are the equivalence classes of the set of terms generated by the following inference rules

$$\frac{t : a \to b \in \mathbf{C}}{t : a \to b \in T(\mathbf{C})} \qquad \frac{t : a \otimes c \to b \otimes c \in T(\mathbf{C})}{tr^c_{a,b}(t) : a \to b \in T(\mathbf{C})}$$

$$\frac{s : a \to b, t : b \to c \in T(\mathbf{C})}{s ; t : a \to c \in T(\mathbf{C})} \qquad \frac{s : a \to b, t : c \to d \in T(\mathbf{C})}{s \oplus t : a \otimes c \to b \otimes d \in T(\mathbf{C})}$$

with respect to the set of axioms in Definition 3.1 and in Definition 3.3, requiring that the operators ; and $\oplus$ are compatible with the operators $*$ and $\otimes$ in $\mathbf{C}$, respectively. We leave to the reader the check that $T(\mathcal{C})$ is indeed a traced gs-monoidal category, and that it verifies the universal property. $\square$

For a given signature $\Sigma$, we call $GS(\Sigma)$ its *gs-monoidal theory*, and we denote it by **GS-Th**$(\Sigma)$; similarly, $T(GS(\Sigma))$ is the *traced gs-monoidal theory* of $\Sigma$, and it is denoted by **TGS-Th**$(\Sigma)$.

A *computad* [57] is a category where each hom-set is equipped with a directed graph structure, whose edges can be regarded as cells which are not closed under composition, in general. The main fact for our analysis is that from a computad a (structured) 2-category can be generated in a free way, by closing the cells under all relevant operations.

**Definition 3.6** (computads). A *gs-monoidal computad* $\mathcal{C}_S$ is a pair $\langle \mathcal{C}, S \rangle$, where $\mathcal{C}$ is a gs-monoidal category and $S$ is a set of cells, each of which has two parallel arrows of $\mathbf{C}$ as *source* and *target*, respectively. A *gs-monoidal c-morphism* $\langle F, h \rangle$ : $\mathcal{C}_S \to \mathcal{D}_T$ is a pair such that $F : \mathcal{C} \to \mathcal{D}$ is a gs-monoidal functor and $h : S \to T$ is a function preserving source and target of cells in the expected way. GS-monoidal computads and their morphisms form a category, denoted **GS-Comp**.

**Proposition 3.7** (free traced gs-monoidal 2-categories). *Let $V_2 :$ **TGS-2Cat** $\rightarrow$ **GS-Comp** be the forgetful functor mapping a traced gs-monoidal 2-category to the underlying gs-monoidal computad (forgetting cell composition and trace structure): it admits a left adjoint $TGS_2 :$ **GS-Comp** $\rightarrow$ **TGS-2Cat**.*

*Proof.* Let $\mathcal{C}_S$ be a gs-monoidal computad, and let $T(\mathcal{C}) = \langle T(\mathbf{C}), \oplus, e, \rho, \nabla, !, tr \rangle$ be the traced gs-monoidal category associated to $\mathcal{C}$. Then the traced gs-monoidal 2-category associated to $\mathcal{C}_S$ is the structure $TGS_2(\mathcal{C}_S) = \langle \underline{\mathbf{C}}, \otimes, e, \rho, \nabla, !, \overline{tr} \rangle$ where the 2-objects and arrows of $\underline{\mathbf{C}}$ are the same as the objects and arrows of $T(\mathbf{C})$, respectively, while the cells are the equivalence classes of the set of terms generated by the following inference rules

$$\frac{t : a \rightarrow b \in T(\mathbf{C})}{id_t : t \Rightarrow t \in \underline{\mathbf{C}}[a, b]} \qquad \frac{\alpha : s \Rightarrow t : a \rightarrow b \in S}{\alpha : s \Rightarrow t \in \underline{\mathbf{C}}[a, b]}$$

$$\frac{\alpha : s \Rightarrow t \in \underline{\mathbf{C}}[a, b], \beta : u \Rightarrow v \in \underline{\mathbf{C}}[b, c]}{\alpha * \beta : s; u \Rightarrow t; v \in \underline{\mathbf{C}}[a, c]} \qquad \frac{\alpha : s \Rightarrow u \in \underline{\mathbf{C}}[a, b], \beta : u \Rightarrow t \in \underline{\mathbf{C}}[a, b]}{\alpha \cdot \beta : s \Rightarrow t \in \underline{\mathbf{C}}[a, b]}$$

$$\frac{\alpha : s \Rightarrow t \in \underline{\mathbf{C}}[a, b], \beta : u \Rightarrow v \in \underline{\mathbf{C}}[c, d]}{\alpha \otimes \beta : s \oplus u \Rightarrow t \oplus v \in \underline{\mathbf{C}}[a \oplus c, b \oplus d]} \qquad \frac{\alpha : s \Rightarrow t \in \underline{\mathbf{C}}[a \oplus c, b \oplus c]}{\overline{tr}_{a,b}^c(\alpha) : tr_{a,b}^c(s) \Rightarrow tr_{a,b}^c(t) \in \underline{\mathbf{C}}[a, b]}$$

with respect to the set of axioms in Definition 3.1 and in Definition 3.3. We leave to the reader the check that $TGS_2(\mathcal{C}_S)$ is indeed a traced gs-monoidal 2-category, and that it verifies the universal property. $\square$

Intuitively, $TGS_2$ generates new cells from those of the argument computad by closing them under all operations of traced monoidal categories and vertical composition, imposing the axioms of traced 2-categories. In addition, it must preserve the gs-monoidal structure on the underlying category, ensuring its compatibility with the analogous operations it introduces for cells.

## 4. REWRITING SEQUENCES AS CELLS OF A 2-THEORY

This section presents the main result of the paper. We first show how to generate a traced gs-monoidal 2-category, denoted **TGS-2Th**$(\mathcal{R})$, from a given rewriting system $\mathcal{R}$; next, we prove that the cells of **TGS-2Th**$(\mathcal{R})$ faithfully correspond to sequences of $\Sigma$-injective rewrites, obtained using the rules of $\mathcal{R}$. In order to give an effective presentation of such a traced gs-monoidal 2-category, we exploit the fact that term graphs over $\Sigma$ can be equipped with a traced gs-monoidal structure, and actually there is an isomorphism between the resulting category and the traced gs-monoidal theory built over $\Sigma$ (see Prop. 3.5).

**Definition 4.1** (the category of term graphs). Let $\Sigma$ be a (one-sorted) signature. **TG**$_\Sigma$ denotes the structure $\langle TG_\Sigma, \oplus, \underline{0}, G_{\Pi(\_,\_)}, G_{\nabla(\_)}, G_{\underline{!}}, tr\rangle$, where

- $TG_\Sigma$ is the category having as objects underlined natural numbers, and as arrows from $\underline{j}$ to $\underline{i}$ all $(i,j)$-ranked term graphs. Arrow composition is defined as term graph composition (Def. 1.4), and the identity on $\underline{i}$ is the discrete term graph $G_{id}^i$ of rank $(i,i)$ having $i$ nodes, where the $k$-th root is also the $k$-th variable, for all $k \in \underline{i}$.
- Functor $\oplus : \mathbf{TG}_\Sigma \times \mathbf{TG}_\Sigma \to \mathbf{TG}_\Sigma$ is defined on arrows as the union of term graphs (Def. 1.4), and on objects as $\underline{n} \oplus \underline{m} = \underline{n+m}$;
- For each $n, m \in \mathbb{N}$, $G_{\Pi(\underline{n},\underline{m})}$ is the discrete term graph of rank $(n+m, n+m)$ with $n + m$ nodes, such that (denoting by $r$ and $v$ the root and variable functions of one of its representatives) $r(x) = v(n + x)$ if $x \leq m$, and $r(x) = v(x - m)$ if $m < x \leq n + m$.
- For each $n \in \mathbb{N}$, $G_{\nabla(\underline{n})}$ is the discrete term graph of rank $(2n, n)$ with $n$ nodes, such that $r(x) = x$ if $x \leq n$, and $r(x) = x - n$ if $n < x \leq 2n$.
- For each $n$, $G_{\underline{!}}^n$ is the discrete term graph of rank $(0, n)$ with $n$ nodes.
- For each $n, m, k \in \mathbb{N}$, the function $tr_{\underline{n},\underline{m}}^k : TG_\Sigma[\underline{n+k}, \underline{m+k}] \to TG_\Sigma[\underline{n},\underline{m}]$ maps a term graph $G$ of rank $(n + k, m + k)$ to $G^{\uparrow^k}$ (see Def. 1.4).

Moreover, **ATG**$_\Sigma$ denotes the structure $\langle ATG_\Sigma, \oplus, \underline{0}, G_{\Pi(\_,\_)}, G_{\nabla(\_)}, G_{\underline{!}}\rangle$, obtained from **TG**$_\Sigma$ by eliminating the trace component $tr$, and where $ATG_\Sigma$ is the subcategory of $TG_\Sigma$ having the same objects but only acyclic term graphs having no $\perp$-nodes as arrows.

**Lemma 4.2** (categories of acyclic term graphs as theories). *Let $\Sigma$ be a (one-sorted) signature. The structure* **ATG**$_\Sigma$ *introduced in Definition 4.1 is a gs-monoidal category isomorphic to* **GS-Th**$(\Sigma)$.

*Proof.* See [15], Theorem 23. $\square$

**Proposition 4.3** (categories of term graphs as theories). *Let $\Sigma$ be a (one-sorted) signature. The structure* **TG**$_\Sigma$ *introduced in Definition 4.1 is a traced gs-monoidal category isomorphic to* **TGS-Th**$(\Sigma)$.

*Proof.* See [19], Theorem 6.2.1. Our traced gs-monoidal categories concide with the $b\delta$-ssmc structures of [19], but for the axiom $tr_{e,e}^1(id_{\underline{1}}) = id_e$, which is not valid in our presentation, since it results in the term graph $[\langle \emptyset, \underline{1}, \emptyset \rangle] : \underline{0} \to \underline{0}$. Nevertheless, the proof there can be carried out smoothly within our setting with minor changes. $\square$

The traced gs-monoidal 2-category generated by a rewriting system over a signature $\Sigma$ will have **TGS-Th**$(\Sigma)$ as underlying category. The result just presented allows us to use term graphs over $\Sigma$ to denote the arrows of such a category, and to make free use of its equational presentation when reasoning about them. Now we first show how to associate (in a quite straightforward way) with each rewriting system $\mathcal{R}$ its *tgs-monoidal 2-theory* **TGS-2Th**$(\mathcal{R})$, and then we proceed by relating cells in this 2-category and rewriting sequences over $\mathcal{R}$.

**Definition 4.4** (rules as cells). Let $\mathbf{R} = \langle d, r_l, r_r \rangle$ be a rule, as in Definition 2.1. Let $n$ be the number of empty nodes of $d$, and let $m$ be the number of nodes of $d|r_l$. Then we define the term graphs $L_\mathbf{R}$ and $R_\mathbf{R}$ of rank $(m, n)$ as follows:

- $L_\mathbf{R} = [\langle r, d|r_l, v \rangle]$, where $r : \underline{m} \to N(d|r_l)$ is a bijection, and $v : \underline{n} \to N_\emptyset(d|r_l)$ is a bijection between $\underline{n}$ and the set of empty nodes of $d|r_l$[5].
- $R_\mathbf{R}$ is defined as the term graph obtained by applying rule $\mathbf{R}$ to $L_\mathbf{R}$ according to Definition 2.2, i.e., $L_\mathbf{R} \to_{\langle \mathbf{R}, id_{d|r_l} \rangle} R_\mathbf{R}$.

The *cell representing rule* $\mathbf{R}$ is the cell $\alpha(\mathbf{R}) : L_\mathbf{R} \Rightarrow R_\mathbf{R} : \underline{n} \to \underline{m}$.

Thus, $R_\mathbf{R}$ is the term graph $[\langle r', d', v \rangle]$, where $d'$ is obtained from $d$ by removing the node $r_l$ and by "redirecting" all pointers from $r_l$ to $r_r$, and $r'(x) = r_r$ if $r(x) = r_l$, and $r'(x) = r(x)$ otherwise. If there is a path in $d$ from $r_r$ to $r_l$, then $d'$ is a cyclic graph (the path must be proper because by definition $r_l \neq r_r$). Indeed, it is easy to see that in this situation any application of the rule results in a cyclic graph: this phenomenon is called *redex capturing* and it is analysed in [30].

**Definition 4.5** (the tgs-monoidal 2-theory of a rewriting system). Let $\mathcal{R} = \langle \{\mathbf{R}_i\}_{i \in I}, \Sigma \rangle$ be a rewriting system, and let $\mathbf{GS\text{-}Th}(\Sigma)$ be the gs-monoidal theory of $\Sigma$. The *gs-monoidal computad representing* $\mathcal{R}$ is defined as the computad $\mathbf{C}_\mathcal{R} = \langle \mathbf{GS\text{-}Th}(\Sigma), \{\alpha(\mathbf{R}_i)\}_{i \in I} \rangle$, where for each rule $\mathbf{R}_i$ in $\mathcal{R}$ the cell $\alpha(\mathbf{R}_i) : L_{\mathbf{R}_i} \Rightarrow R_{\mathbf{R}_i}$ is as in Definition 4.4. The *tgs-monoidal 2-theory of* $\mathcal{R}$, denoted by $\mathbf{TGS\text{-}2Th}(\mathcal{R})$, is the free traced gs-monoidal 2-category generated by $\mathbf{C}_\mathcal{R}$, i.e., $TGS_2(\mathbf{C}_\mathcal{R})$, where $TGS_2$ is as in Proposition 3.7.

Before presenting our main result, we need two technical lemmas, relating $\Sigma$-injective morphisms and term graph *contexts*. These are obvious generalisations of the corresponding results for the acyclic case, presented in the full version of [13].

**Definition 4.6** (term graph contexts). A *(term graph) context* $C[i, j]$ is a well-formed term graph expression containing exactly one (ranked) place-holder $[i, j]$ (for some $i, j \in \mathbb{N}$).

We write $C[i, j] : (n, m)$ to say that the rank of $C[i, j]$ is $(n, m)$, assuming that the place-holder has rank $(i, j)$. If $C[i, j]$ is a context and $G$ is a term graph of rank $(i, j)$, by $C[G]$ we denote the term graph obtained by evaluating the expression $C$ after replacing $[i, j]$ by $G$.

**Lemma 4.7** ($\Sigma$-injective morphisms and contexts). *Let $G$ be an acyclic term graph of rank $(m, n)$ without $\perp$-nodes such that all its nodes are roots, and let $H$ be a term graph.*

1. *If $C[m, n]$ is a context such that $H = C[G]$, then for every $g_G \in G$ and $g_H \in H$ a $\Sigma$-injective graph morphism $\phi(C) : d_G \to d_H$ between the underlying graphs can be defined inductively on the structure of $C$.*

---

[5]Note that $r$ and $v$ are well-defined by the definition of $n$ and $m$; moreover, they could be fixed in a canonical way, but we do not need this.

2. *If $g_G \in G$ and $g_H \in H$, then for every $\Sigma$-injective graph morphism $f : d_G \to d_H$ between the underlying graphs a context $\gamma(f)[m, n]$ can be identified such that $\gamma(f)[G] = H$; furthermore, $\phi(\gamma(f)) = f$.*

*Proof.* (1) Since the proof goes by induction on the structure of the context, we may assume that the base case (note that $[G] = G$ by definition, and all the nodes are also roots, thus forcing a unique choice for the graph isomorphism) and the other inductive steps are already verified [13], except for $C[m, n] = (C'[m, n])^\dagger$. So, let $g_{H'}$ be a chosen graph in $H' = C'[G]$ such that $(g_{H'})^\dagger$ is isomorphic to $g_H$ via a morphism $h$ (such a choice is always possible, since we defined constructively the operation of feedback: see the remark after Definition 1.4), and let $\phi(C') : d_G \to d_{H'}$ be the $\Sigma$-injective graph morphism identified by induction hypothesis. Then, the morphism $\phi(C) = h \circ u_{d_{H'}} \circ \phi(C') : d_G \to d_H$ satisfies the required conditions, for $u_{d_{H'}} : d_{H'} \to (d_{H'})^\dagger$ the $\Sigma$-injective graph morphism induced by the construction in Definition 1.4.

(2) We can assume that the statement holds for any graph $g_H$ containing no cycle and no $\perp$-node [13]. Thus, the proof proceeds similarly to the proof of Proposition 1.8. Let us consider *e.g.* a graph $g_{H'}$ containing one cycle less than $g_H$, such that $g_H$ is isomorphic to $(g_{H'})^\dagger$ via a morphism $h$, and such that $f$ induces an arrow $f' : d_G \to d_{H'}$ satisfying $h \circ u_{d_{H'}} \circ f' = f$ for the $\Sigma$-injective graph morphism $u_{d_{H'}} : d_{H'} \to (d_{H'})^\dagger$. Note that the choice of the cycle in $g_H$ to be reduced (hence, the choice of $g_{H'}$) is not determined uniquely, while the subsequent choice of $h$ is. By induction hypothesis, $\gamma(f')[G] = H'$ and $\phi(\gamma(f')) = f'$. We then define $\gamma(f) = (\gamma(f'))^\dagger$. By the previous point, $\phi((\gamma(f'))^\dagger) = h \circ u_{d_{H'}} \circ \phi(\gamma(f'))$, so that $\phi(\gamma(f)) = \phi((\gamma(f'))^\dagger) = h \circ u_{d_{H'}} \circ \phi(\gamma(f')) = f$.    □

Please note that the above correspondence between contexts and morphisms (in the same equivalence class) is not one-to-one, since we lack an explicit equivalence relation equating those contexts intuitively corresponding to the same arrow.

**Lemma 4.8** (term graph rewriting as contextualisation). *Let $\mathbf{R} = \langle d, r_l, r_r \rangle$ be a rule and $\alpha : L_{\mathbf{R}} \Rightarrow R_{\mathbf{R}} : \underline{n} \to \underline{m}$ the associated cell by Definition 4.4. Then for every context $C[m, n]$ and for every graph $g_0$ in $C[L_{\mathbf{R}}]$ the pair $\Delta = \langle \mathbf{R}, \phi(C) : d|r_l \to d_0 \rangle$ is a $\Sigma$-injective redex of $C[L_{\mathbf{R}}]$ such that $C[L_{\mathbf{R}}] \to_\Delta C[R_{\mathbf{R}}]$.*

*Proof.* Let $\phi(C) : d|r_l \to d_0$ be the $\Sigma$-injective graph morphism whose characterisation is ensured by Lemma 4.7: we just need to prove that $C[L_{\mathbf{R}}] \to_{\langle \mathbf{R}, \phi(C) \rangle} C[R_{\mathbf{R}}]$. As before, since the proof goes by induction on the structure of the context, we may assume that the base case (note that $\phi([m, n]) = id$, and $L_{\mathbf{R}} \to_{\langle \mathbf{R}, id \rangle} R_{\mathbf{R}}$ by definition) and the other inductive steps are already verified, except for $C[m, n] = (C'[m, n])^\dagger$. So, let $g_0'$ be a chosen graph in $C'[L_{\mathbf{R}}]$, such that $(g_0')^\dagger$ is isomorphic to $g_0$ *via* a morphism $h$, and let $\phi(C') : d|r_l \to d_0'$ be the unique arrow induced by Lemma 4.7, so that by induction hypothesis $C'[L_{\mathbf{R}}] \to_{\langle \mathbf{R}, \phi(C') \rangle} C'[R_{\mathbf{R}}]$.

Let us assume that $g_0' \to_{\langle \mathbf{R}, \phi(C') \rangle} g_1'$: it is not difficult to check that the application of the algorithm in Definition 2.2 transforms $(g_0')^\dagger$ into $(g_1')^\dagger$, hence $C[L_{\mathbf{R}}] \to_{\langle \mathbf{R}, \phi(C) \rangle} C[R_{\mathbf{R}}]$, and the result holds[6]. □

We are now ready to present our main result: its proof follows closely the one presented in [13] for acyclic rewriting, exploiting the two previous lemmas.

**Theorem 4.9** (rewriting sequences as cells of the traced gs-monoidal 2-theory).
*Let $\mathcal{R}$ be a rewriting system, and let $G$ and $H$ be two term graphs having the same rank $(m, n)$. Then there is a cell $\alpha : G \Rightarrow H : \underline{n} \to \underline{m}$ in $\mathbf{TGS\text{-}2Th}(\mathcal{R})$ if and only if there is a $\Sigma$-injective rewriting sequence $G \to_{\mathcal{R}}^* H$.*

*Proof.* We first ask the reader to look back at the construction of the free traced gs-monoidal 2-category we presented in the proof of Proposition 3.7, and to note that the cells in $\mathbf{TGS\text{-}2Th}(\mathcal{R})$ are generated from the cells of computad $\mathbf{C}_{\mathcal{R}}$ (the *2-generators*) by closing them with respect to all the operations of traced gs-monoidal categories, and by imposing the corresponding axioms. Using structural induction, a cell $\alpha$ can be proved decomposable into an equivalent (even if not necessarily uniquely defined) cell $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_k$ (where "·" denotes (vertical) composition in the hom-category $\mathbf{TGS\text{-}2Th}(\mathcal{R})(\underline{n}, \underline{m})$) such that each $\alpha_i$ contains exactly one 2-generator and no vertical composition. Moreover, each single cell $\alpha : G \Rightarrow H$ containing only one 2-generator can be further decomposed as $C[\alpha(\mathbf{R})]$, for a suitable context $C$ and a rule $\mathbf{R} \in \mathcal{R}$.

*Only if part.* Let us restrict our attention to a single cell $\alpha : G \Rightarrow H$ containing only one 2-generator, and let assume that it can be decomposed as $C[\alpha(\mathbf{R})]$ for a suitable context $C$ and a rule $\mathbf{R} \in \mathcal{R}$. By the definition of $\alpha(\mathbf{R})$, we have that $G = C[L_{\mathbf{R}}]$ and $H = C[R_{\mathbf{R}}]$, and by Lemma 4.8 $G \to_{\langle \mathbf{R}, \phi(C) \rangle} H$. The statement follows by observing that vertical composition directly corresponds to concatenation of rewriting sequences.

*If Part.* If $G \to_\Delta H$ using a $\Sigma$-injective redex $\Delta = \langle \mathbf{R}, f \rangle$ with $\mathbf{R} \in \mathcal{R}$, then by Lemma 4.7 there is a context $\gamma(f)$ such that $G = \gamma(f)[L_{\mathbf{R}}]$. By Definition 4.4 and Definition 4.5, the cell $\alpha(\mathbf{R}) : L_{\mathbf{R}} \Rightarrow R_{\mathbf{R}}$ is a 2-generator of $\mathbf{TGS\text{-}2Th}(\mathcal{R})$, and thus, since cells are closed under arbitrary contexts, $\gamma(f)[\alpha(\mathbf{R})] : G \Rightarrow \gamma(f)[R_{\mathbf{R}}]$ is a cell. It remains to show that $H = \gamma(f)[R_{\mathbf{R}}]$. By Lemma 4.8 we have $G \to_{\langle \mathbf{R}, \phi(\gamma(f)) \rangle} \gamma(f)[R_{\mathbf{R}}]$, and by Lemma 4.7 $\phi(\gamma(f)) = f$, so that $H = \gamma(f)[R_{\mathbf{R}}]$ by the determinacy of the rewriting algorithm in Definition 2.2. The statement easily extends to arbitrary rewriting sequences using vertical composition. □

## 5. Discussion

Our definition of term graph rewriting (Def. 2.2) differs from the corresponding definition in [3] for two main aspects: The presence of an initial

---

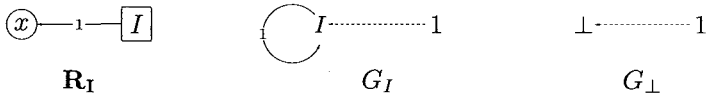[6]Informally, the rewrite could be affected by the presence of the external $(-)^\dagger$ context only if the feedback should involve the root $r_l$. In that case, it is easy to see that the *Root undefine phase* guarantees the correctness, should the image of the left and right roots coincide: it severs the link outgoing from the left root, even if the node itself is glued with the right root.

phase (*root undefine*) where we transform the root of the redex in an empty node; and the absence of a final *garbage collection phase*, removing from the resulting term graph all the non-variable nodes not reachable from the roots. We first discuss these two facts in turn, then we try to sustain our claim that the categorical framework we presented allows for an easy comparison between various instances of term (graph) rewriting. We conclude the section with a short historical overview on the algebraic approach to the notion of fixed point and feedback.

## 5.1. THE "ROOT UNDEFINE PHASE" AND HANDLING OF CIRCULAR REDEXES

As far as the presence of the initial *Root undefine phase* in Definition 2.2 is concerned, it is easy to check that it affects the result of the rewriting only in the case of *circular redexes*, that is, when the image of the left and of the right roots of the rule coincide. For $\Sigma$-injective redexes, this can only happen with *collapsing rules*, *i.e.*, those where the right root is a variable. The paradigmatic example here is the *circular I*, *i.e.*, the application of the identity rule $\mathbf{R_I} : I(x) \to x$ to the cyclic graph $G_I$ having one node labelled $I$ and one looping edge.

$$\boxed{x}\!\!-\!\!1\!\!-\!\!\boxed{I} \qquad\qquad \mathrlap{\text{(}}I\cdots\cdots 1 \qquad\qquad \perp\cdots\cdots 1$$

$$\mathbf{R_I} \qquad\qquad\qquad\qquad G_I \qquad\qquad\qquad G_\perp$$

Using the definition in [3] or, equivalently, our Definition 2.2 without the first phase, this redex reduces to itself. In fact, since the image of the left root $(I)$ and of the right root $(x)$ of $\mathbf{R_I}$ coincide in $G_I$, the Build, Redirect and Root removal phases have no effect on $G_I$. Instead, applying the whole construction of Definition 2.2, the Root undefine phase makes the $I$-node of $G_I$ empty, and the other phases leave it unchanged. Thus the result is graph $G_\perp$ consisting of a single $\perp$-node which is the only root, *i.e.*, $G_I \to_{\mathbf{R_I}} G_\perp$. Another example of reduction of a circular redex is the rewriting $G_3 \to_{\mathbf{R_2}} G_4$ at the end of Section 1.

It is worth mentioning that other definitions of term graph rewriting agree with ours on circular redexes (see $[1, 12, 34, 40]^7$), and that many authors already discussed the disagreement of various definitions of rewriting in the handling of circular redexes. In our opinion, this disagreement is due to the operational nature of most definitions, which leaves some degree of freedom in the handling of "pathological" cases like circular redexes. To our knowledge, our paper is the first one where an operational construction is proved to be correct with respect to a more declarative, categorical counterpart, as stated by the main result, Theorem 4.9.

It is instructive to determine the cell of $\mathbf{TGS\text{-}2Th}(\{I\})$ which corresponds to the reduction of the circular $I$. According to Definition 4.4, the cell representing $\mathbf{R_I}$ is $\alpha(\mathbf{R_I}) = (\nabla_{\underline{1}} * (id_{\underline{1}} \otimes I)) \Rightarrow \nabla_{\underline{1}} : \underline{1} \to \underline{2}$, and since cells are closed under traces, there is a cell $tr^1_{\underline{0},1}(\alpha(\mathbf{R_I})) = tr^1_{\underline{0},1}(\nabla_{\underline{1}} * (id_{\underline{1}} \otimes I)) \Rightarrow tr^1_{\underline{0},1}(\nabla_{\underline{1}}) : \underline{0} \to \underline{1}$. It is easy to check that the source of this cell represents the term graph $G_I$, while the

---

[7]In fact, our definition of term graph rewriting coincides with that proposed in [12], which is based on the "Double-Pushout Approach" to graph rewriting. We refrained from using it, in order to ease the comparison with [3].

target is $G_\perp$. Informally, we can say that a circular redex reduces to $\perp$ because the categorical structure forces the rewrite relation to be a *pre-congruence*, that is, to be closed with respect to all the operators, including feedback. This shows that our main result does not hold if circular redexes are treated in a different way in the operational construction.

## 5.2. ADDING GARBAGE COLLECTION TO TERM GRAPH REWRITING

The absence of garbage collection in our definition is due to the fact that, on the categorical side, the 2-transformation "!" is not required to be natural. As argued in [15], the naturality of ! is the categorical counterpart of *automatic garbage collection*, since it implies $g*!_{\underline{1}} = !_{\underline{n}} : \underline{n} \to \underline{0}$ for any $n$-ary operator $g$, meaning that a term graph with a single garbage node $g$ is "the same" as the term graph obtained by removing it. Also, for each term graph $G$ of rank $(1, n+1)$ we would have $tr^1_{\underline{n},\underline{0}}(G) = tr^1_{\underline{n},\underline{0}}(G)*!_{\underline{0}} = !_{\underline{n}}$ (because $!_{\underline{0}} = id_{\underline{0}}$), meaning that all garbage nodes can be safely removed at once from a cyclic term graph without roots. As a consequence, we obtain a categorical representation (analogous to Th. 4.3) of "term graphs without garbage nodes", and a correspondence between $\Sigma$-injective rewrites *with* garbage collection and cells of the 2-theory, as in Theorem 4.9, simply by requiring the naturality of ! in the definition of traced gs-monoidal 2-category.

The categorical framework makes easy also to consider an alternative solution, consisting of adding a set of "garbage collection rules" which incrementally delete the garbage nodes produced during the rewriting. For this, it is sufficient to require that "!" is an *op-lax natural* 2-transformation.

**Definition 5.1** (lax, op-lax, and pseudo-natural 2-transformations [39]). Let $F, G : \mathbf{C} \to \mathbf{D}$ be 2-functors. A 2-transformation $\eta : F \Rightarrow G$ is *lax natural* if it comes equipped with a family of cells $\eta$ of $\mathbf{D}$, indexed by arrows $f \in \mathbf{C}[a, b]$, such that $\eta_f : \eta_a * G_{a,b}(f) \Rightarrow F_{a,b}(f)*\eta_b \in \mathbf{D}[F_o(a), G_o(b)]$; it is *op-lax natural* if it comes equipped with a family of cells $\eta^{co}$ as above, but in the opposite direction, *i.e.*, $\eta^{co}_f : F_{a,b}(f) * \eta_b \Rightarrow \eta_a * G_{a,b}(f)$; it is *pseudo-natural* if it is both lax and op-lax natural, and for each arrow $f$ the cells $\eta_f$ and $\eta^{co}_f$ form an isomorphic pair.

**Theorem 5.2** (rewriting sequences with garbage collection as cells). *Let* $\mathcal{R} = \langle \{\mathbf{R}_i\}_{i \in I}, \Sigma \rangle$ *be a rewriting system, and let* $\mathbf{TGS\text{-}2Th}_{gc}(\mathcal{R})$ *be the free traced gs-monoidal 2-category generated by the gs-computad* $\langle \mathbf{GS\text{-}Th}(\Sigma), S \rangle$*, requiring "!" to be an* op-lax natural *transformation.*

*Now let* $G$ *and* $H$ *be two term graphs with no garbage nodes. Then there is a cell* $\alpha : G \Rightarrow H$ *in* $\mathbf{TGS\text{-}2Th}_{gc}(\mathcal{R})$ *if and only if there is a $\Sigma$-injective rewriting sequence* $G \to^*_{gc,\mathcal{R}} H$*, where the subscript* gc *means that we add after the* root removal *phase a garbage collection* one.

The result above follows easily from Theorem 4.9, by observing that the additional rewrites can only be applied to garbage nodes. The op-lax naturality breaks down to the presence of cells $g*!_{\underline{1}} \Rightarrow !_{\underline{n}}$ and $tr^1_{\underline{n},\underline{0}}(G) \Rightarrow !_{\underline{n}}$ for each n-ary operator $g$ and each graph $G$ of rank $(1, n+1)$. Therefore, in this formulation garbage collection can be executed concurrently with other rewrites, without

affecting the correctness of the system. In fact, our solution closely resembles the classical *reference counting* algorithm for distributed garbage collection, where each processor (that is, each node) is equipped with the number of other processors actually linked to it: while $g*!_{\underline{1}}$ means that no pointers is looking up to $g$, and thus it can be deleted, $tr^1_{\underline{n},0}(G)$ can be considered as an oracle, statically detecting the occurrence of a self-reference cycle, and allowing for its deletion.

## 5.3. REWRITING WITH FOLDING/UNFOLDING AND SHARING/UNSHARING

Theorem 4.9 states that cells in the 2-theory of $\mathcal{R}$ represent $\Sigma$-*injective* rewriting sequences. This restriction can be relaxed by adding to the system explicit rules which change locally the degree of sharing in a term graph (for the acyclic case, such rules have been considered *e.g.* in [2, 36]). Categorically, this is obtained requiring the transformation "$\nabla$" to be *pseudo-natural*. Then, for each $n$-ary operator $f$, the cell $\nabla_f : f * \nabla_{\underline{1}} \Rightarrow \nabla_{\underline{n}} * (f \otimes f)$ allows one to create two copies of $f$ if it is shared, while the inverse $\nabla^{co}_f : \nabla_{\underline{n}} * (f \otimes f) \Rightarrow f * \nabla_{\underline{1}}$ merges together two copies of $f$ if they share all their successors. Furthermore, derived cells allow one to "unfold" or "fold" cyclic structures. For example, let us consider again the term $tr^1_{\underline{0},1}(\nabla_{\underline{1}} * (id_{\underline{1}} \otimes I))$, representing the "$I$-loop" $G_I$: it is equivalent to $tr^1_{\underline{0},1}(I * \nabla_{\underline{1}})$, by the first naturality axiom of Definition 3.3. By pseudo-naturality of $\nabla$, and since cells are closed under traces, there is a cell $tr^1_{\underline{0},1}(\nabla_I) : tr^1_{\underline{0},1}(I * \nabla_{\underline{1}}) \Rightarrow tr^1_{\underline{0},1}(\nabla_{\underline{1}} * (I \otimes I))$, and this last term is equivalent to $tr^1_{\underline{0},1}(\nabla_{\underline{1}} * (id_{\underline{1}} \otimes I)); I$, by the second naturality axiom of Definition 3.3: it then represents a partial unfolding of the $I$-loop.

**Theorem 5.3** (rewriting sequences with unsharing as cells). *Let* $\mathcal{R} = \langle \{\mathbf{R}_i\}_{i \in I}, \Sigma \rangle$ *be a rewriting system, and let* $\mathbf{TGS\text{-}2Th}_s(\mathcal{R})$ *be the free traced gs-monoidal 2-category generated by the gs-computad* $\langle \mathbf{GS\text{-}Th}(\Sigma), S \rangle$, *requiring* "$\nabla$" *to be a* pseudo-natural *transformation.*

*Now let* $G$ *and* $H$ *be two term graphs such that there is a rewriting sequence* $G \to^*_{\mathcal{R}} H$: *Then there exists a cell* $\alpha : G \Rightarrow H$ *in* $\mathbf{TGS\text{-}2Th}_s(\mathcal{R})$. $\qquad\square$

In the extended 2-theory $\mathbf{TGS\text{-}2Th}_s(\mathcal{R})$ a rewriting step *via* a non-$\Sigma$-injective redex can be simulated by a cell composed of a sequence of unsharing/unfolding cells (that transform the redex into a $\Sigma$-injective one), of the 2-generator corresponding to the rule, and finally of a sequence of sharing/folding cells (that restore the original degree of sharing). For acyclic graphs, the theorem is worked out in detail in the full version of [13], and it is closely related to Theorem 7.1 in [2], which is not based on a categorical framework. In the general case, the result relies on the interweaving between trace operators and sharing/unsharing rules, modelling the process of folding/unfolding of cycles. The converse of the theorem clearly does not hold, because in $\mathbf{TGS\text{-}2Th}_s(\mathcal{R})$ more "housekeeping" operations can be performed than it is actually required for the application of a given rule.

## 5.4. RELATING REWRITING FORMALISMS VIA ADJUNCTIONS

The categorical presentation can be exploited to relate formalisms *via* suitable adjunctions. The diagram below summarises the relationships between traced gs-monoidal 2-categories and other 2-categories used to represent other rewriting formalisms based on terms or term graphs. Arrows represent adjunctions in the direction of the left adjoint. Adjunction (1) essentially models the 2-categorical presentation of term rewriting using algebraic 2-theories [44,49,51], where a rewriting system is represented by a *cartesian* computad (an element of **C-Comp**), and its free cartesian 2-category has cells for all rewriting sequences. Adjunction (2) models acyclic term graph rewriting, and it is described in [13], where also the commutativity of the top square of adjunctions is discussed. Adjunction (3) models *rational term rewriting* (RTR), *i.e.*, the extension of standard term rewriting obtained by allowing for possibly infinite terms with a finite number of sub-terms, and for *infinite parallel rewriting*, as originally defined in [11]. Both an operational and a logical presentations of RTR are proposed in [14], and in the corresponding full paper we plan to include the categorical presentation of RTR based on adjunction (3) and on *iteration 2-theories* [9, 28]. Adjunction (4) models possibly cyclic term graph rewriting, as described in this paper. The vertical adjunctions (7,8) add traced or feedback structure, resulting in possibly cyclic term graphs to the left, and in rational terms to the right. Finally, the horizontal adjunctions (0,5,6) are obtained by enforcing the naturality of $\nabla$ and $!$[8].
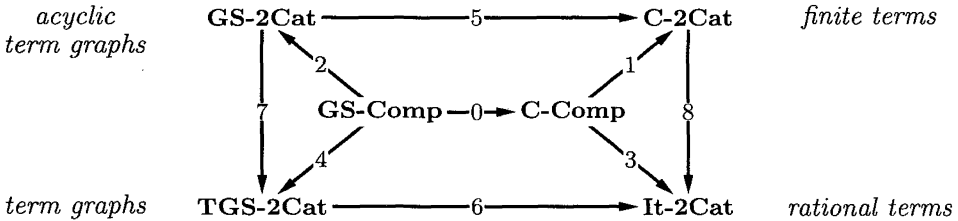
In particular, adjunction (6) maps the traced gs-monoidal theory of a signature **TGS-Th($\Sigma$)** to the *iteration theory* **It-Th($\Sigma$)**, and the unit of the adjunction maps a term graph $G$ of rank $(n, m)$ to an $n$-tuple of rational terms over $m$ variables, essentially obtained by "unraveling" $G$ from its roots. It is especially interesting to note that this framework helps us to clarify the rôle of $\perp$-nodes of term graphs, *i.e.*, those resulting from the reduction of a circular redex. We argued in [11,12] that such a node corresponds to the completely undefined term, *i.e.*, the bottom element of the CPO of possibly infinite, *possibly partial* terms. This turns out to be correct, because rational terms in the iteration theory of $\Sigma$ are naturally endowed with a CPO structure, and the above mentioned unit maps such nodes

---

[8]Adjunctions (3,6,8) actually need an additional set of axioms, besides those for the trace structure, in order to deal with the occurrence of nested cycles. The first axiomatisation for iteration theories appeared in [26], by means of a set of *commutative identities*. Related solutions include the *group axioms* of the recent [27], or the *functorial implication* (also *enzymatic rule*) used in [19,21,26]. Without these axioms, the resulting structure is a *Conway 2-theory* [4,9,28].

to the bottom element there (which explains why we denoted them by $\perp$). This interpretation compares favourably with the treatment of such nodes in *e.g.* [1], for which a fresh constant • is added to the signature.

## 5.5. SOME HISTORICAL REMARKS

The observation that cartesian categories are monoidal categories equipped with additional natural transformations is quite old, and indeed many authors studied categorical structures that are weaker than the former and richer than the latter. We refer the reader to the concluding remarks of [15] for a survey on some of the motivations leading to such structures. It is instead relatively easier to track down the interest towards (cartesian) categories enriched with enough properties in order to encompass an explicit *recursive* structure, since it can be largely dated back to the first studies on the semantics of *flowchart schemes*, intended as syntactic devices representing the control structure of a program.

Roughly, a flowchart scheme is a term graph (actually, it would be better described as a *cyclic hypergraph*, but we can avoid this distinction in our discussion, for the sake of clarity), where each node represents a functionality of the program, and the links denote the control dependencies between the various components. In this setting, the fixed point operator can be interpreted as a direct translation of the GO TO statement: such an intuition explains the rationale behind the pioneering work of, among others, Elgot and Wand [23,59], where a feedback (or iteration) operator was introduced in order to provide a linear syntax mechanism for the inductive definition of flowchart schemes. For example, Theorem 3.1 of [24] presents a decomposition result for flowchart schemes that subsumes our Proposition 1.8. The connection between algebraic theories and the step-by-step semantics for program schemes moved the interest towards cartesian categories, either order enriched [32,52] or equipped with a (unique) fixed point operator. The privileged status of algebraic theories was further motivated by the correspondence between rational trees and certain varieties of iterative algebras [25,31], thus providing a characterisation for behaviourally equivalent flowchart schemes. This is the line that brought the development of Iteration Theories, as recollected in [7,8].

What we may consider the first results on the completeness of a linear syntax for the presentation of flowchart schemes–and its relationship with their behavioural semantics–are due to Bloom and Èsik [6]. This is the line bringing to the in-depth analysis of such axiomatics represented by the *algebra of flownomials* developed by Căzănescu and Ştefănescu [18,20,21]. Roughly, in their approach they relate flowchart schemes to symmetric monoidal categories enriched with (non-natural) transformations that correspond to our $\nabla$ and !, as well as to their duals, and with an explicit feedback operator. Even if our presentation uses the *trace* structure of [37], introduced to deal with *tortile* monoidal categories, for symmetric monoidal categories it is essentially the same as in [20][9].

---

[9] In fact, our Proposition 4.3 already appeared in the literature, most notably in the mentioned [19]. Also the intuition that "iteration = feedback + cartesianity"–corresponding to the arrow

Term graph rewriting has a long history inside the theoretical computer science field, too. The starting point can be considered the need of efficient implementations for recursive program schemes, linked with their operational interpretation (see *e.g.* the textbook [33] by Guessarian for early references). The syntactical solution proposed by Berry and Lévy [5] (and generalised by Boudol [10]) introduces a notion of *family reduction* over terms which simulates some kind of rewriting up-to sharing. The operational solution instead directly implements a term as a graph, transforming each equation occurring in a program scheme into a graph rewriting rule. While the work of Staples [54–56] may be the first to address explicitly the problem of correctness and completeness for such transformations, it is with [3] that an accepted notation for term graph rewriting is established. It is noteworthy that, given the operational interpretation of the program scheme, fixed points enter the picture only as a side-effect of suitable implementation strategies of the rewriting, as shown by [30], and since then cyclic term graph rewriting has been considered by various authors [1, 12, 29, 41].

To a certain extent, we consider our work at the borderline of both areas. It fits into the term graph rewriting community, since it discusses an alternative solution for the collapsing rules problem. But it also provides a linear syntax presentation for both term graphs (the data structure), and for term graph rewrites (the computational structure), thus offering insights on the relationship between the operational and the syntactical solution to the optimal implementation problem for term rewriting. In order to provide such a correspondence, it recalls previous results (in particular, the equational presentations) from the field of flowchart schemes, thus establishing a connection–that was present in the early Seventies, but has since then been overlooked– between now separated threads of research.

## References

[1] Z.M. Ariola and J.W. Klop, Equational term graph rewriting. *Fund. Inform.* **26** (1996) 207–240.

[2] Z.M. Ariola, J.W. Klop and D. Plump, Confluent rewriting of bisimilar term graphs, C. Palamidessi and J. Parrow, Eds., *Expressiveness in Concurrency*. Elsevier Sciences, *Electron. Notes Theoret. Comput. Sci.* **7** (1997). Available at http://www.elsevier.nl/locate/entcs/volume7.html/.

[3] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep, Term graph reduction, J.W. de Bakker, A.J. Nijman and P.C. Treleaven, Eds., *Parallel Architectures and Languages Europe*. Springer Verlag, *Lecture Notes in Comput. Sci.* **259** (1987) 141–158.

[4] L. Bernátsky and Z. Ésik, Semantics of flowchart programs and the free Conway theories. *Theoret. Informatics Appl.* **32** (1998) 35–78.

[5] G. Berry and J.-J. Lévy, Minimal and optimal computations of recursive programs. *J. ACM* **26** (1979) 148–175.

[6] S. Bloom and Z. Ésik, Axiomatizing schemes and their behaviors. *J. Comput. System Sci.* **31** (1985) 375–393.

component of the adjunction 6 in the diagram of Section 5.4, and reconsidered recently by Hasegawa [35] and Hyland–can be dated back to [6], and explicitly to [4, 20, 22].

[7] S. Bloom and Z. Ésik, *Iteration Theories*. EATCS Monographs on Theoretical Computer Science. Springer Verlag (1993).

[8] S. Bloom and Z. Ésik, The equational logic of fixed points. *Theoret. Comput. Sci.* **179** (1997) 1–60.

[9] S.L. Bloom, Z. Ésik, A. Labella and E.G. Manes, Iteration 2-theories, M. Johnson, Ed., *Algebraic Methodology and Software Technology*. Springer Verlag, *Lecture Notes in Comput. Sci.* **1349** (1997) 30–44.

[10] G. Boudol, Computational semantics of term rewriting systems, M. Nivat and J. Reynolds, Eds., *Algebraic Methods in Semantics*. Cambridge University Press (1985) 170–235.

[11] A. Corradini, Term rewriting in $CT_\Sigma$, M.-C. Gaudel and J.-P. Jouannaud, Eds., *Trees in Algebra and Programming*. Springer Verlag, *Lecture Notes in Comput. Sci.* **668** (1993) 468–484.

[12] A. Corradini and F. Drewes, (Cyclic) term graph rewriting is adequate for rational parallel term rewriting, Technical Report TR-97-14. Dipartimento di Informatica, Pisa (1997).

[13] A. Corradini and F. Gadducci, A 2-categorical presentation of term graph rewriting, E. Moggi and G. Rosolini, Eds., *Category Theory and Computer Science*. Springer Verlag, *Lecture Notes in Comput. Sci.* **1290** (1997) 87–105.

[14] A. Corradini and F. Gadducci, Rational term rewriting, M. Nivat, Ed., *Foundations of Software Science and Computation Structures*. Springer Verlag, *Lecture Notes in Comput. Sci.* **1378** (1998) 156–171.

[15] A. Corradini and F. Gadducci, An algebraic presentation of term graphs, *via* gs-monoidal categories. *Applied Categorical Structures*, to appear.

[16] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel and M. Löwe, Algebraic approaches to graph transformation I: Basic concepts and double pushout approach, G. Rozenberg, Ed., *Handbook of Graph Grammars and Computing by Graph Transformation 1*. World Scientific (1997).

[17] A. Corradini and F. Rossi, Hyperedge replacement jungle rewriting for term rewriting systems and logic programming. *Theoret. Comput. Sci.* **109** (1993) 7–48.

[18] Gh. Ştefănescu, On flowchart theories: Part II. The nondeterministic case. *Theoret. Comput. Sci.* **52** (1987) 307–340.

[19] Gh. Ştefănescu, Algebra of flownomials. Technical Report SFB-Bericht 342/16/94 A. Technical University of München, Institut für Informatik (1994).

[20] V.-E. Căzănescu and Gh. Ştefănescu, Towards a new algebraic foundation of flowchart scheme theory. *Fund. Inform.* **13** (1990) 171–210.

[21] V.-E. Căzănescu and Gh. Ştefănescu, A general result on abstract flowchart schemes with applications to the study of accessibility, reduction and minimization. *Theoret. Comput. Sci.* **99** (1992) 1–63.

[22] V.-E. Căzănescu and Gh. Ştefănescu, Feedback, iteration and repetition, Gh. Paun, Ed., *Mathematical Aspects of Natural and Formal Languages*. World Scientific (1995) 43–62.

[23] C.C. Elgot, Monadic computations and iterative algebraic theories, H.E. Rose and J.C. Shepherdson, Eds., *Logic Colloquium 1973*. North Holland, *Stud. Logic Found. Math.* **80** (1975) 175–230.

[24] C.C. Elgot, Structured programming with and without GO TO statements. *IEEE Trans. Software Engrg.* **2** (1976) 41–54.

[25] C.C. Elgot, C.C. Bloom and R. Tindell, The algebraic structure of rooted trees. *J. Comput. System Sci.* **16** (1978) 362–339.

[26] Z. Ésik, Identities in iterative theories and rational algebraic theories. *Computational Linguistics and Computer Languages* **14** (1980) 183–207.

[27] Z. Ésik, Group axioms for iteration. *Inform. and Comput.* **148** (1999) 131–180.

[28] Z. Ésik and A. Labella, Equational properties of iteration in algebraically complete categories. *Theoret. Comput. Sci.* **195** (1998) 61–89.

[29] W.M. Farmer, J.D. Ramsdell and R.J. Watro, A correctness proof for combinator reduction with cycles. *ACM Trans. Program. Lang. Syst.* **12** (1990) 123–134.

[30] W.M. Farmer and R.J. Watro, Redex capturing in term graph rewriting, R.V. Book, Ed., *Rewriting Techniques and Applications*. Springer Verlag, *Lecture Notes in Comput. Sci.* **488** (1991) 13–24.

[31] S. Ginali, Regular trees and the free iterative theory. *J. Comput. System Sci.* **18** (1979) 222–242.

[32] J.A. Goguen, J.W. Tatcher, E.G. Wagner and J.R Wright, Initial algebra semantics and continuous algebras. *J. ACM* **24** (1977) 68–95.

[33] I. Guessarian, *Algebraic Semantics*. Springer Verlag, *Lecture Notes in Comput. Sci.* **99** (1981).

[34] M. Hasegawa, *Models of Sharing Graphs*. Ph.D. Thesis, University of Edinburgh, Department of Computer Science (1997).

[35] M. Hasegawa, Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda-calculus, Ph. de Groote and R. Hindly, Eds., *Typed Lambda Calculi and Applications*. Springer Verlag, *Lecture Notes in Comput. Sci.* **1210** (1997) 196–213.

[36] B. Hoffmann and D. Plump, Implementing term rewriting by jungle evaluation. *Theoret. Informatics Appl.* **25** (1991) 445–472.

[37] A. Joyal, R.H. Street and D. Verity, Traced monoidal categories. *Math. Proc. Cambridge Philos. Soc.* **119** (1996) 425–446.

[38] G. Kelly, *Basic Concepts of Enriched Category Theory*. Cambridge University Press (1982).

[39] G.M. Kelly and R.H. Street, Review of the elements of 2-categories, G.M. Kelly, Ed., *Sydney Category Seminar*. Springer Verlag, *Lecture Notes in Math.* **420** (1974) 75–103.

[40] J.R. Kennaway, On "On Graph Rewritings". *Theoret. Comput. Sci.* **52** (1980) 37–58.

[41] J.R. Kennaway, J.W. Klop, M.R. Sleep and F.J. de Vries, On the adequacy of graph rewriting for simulating term rewriting. *ACM Trans. Program. Lang. Syst.* **16** (1994) 493–523.

[42] J.W. Klop, Term rewriting systems, S. Abramsky, D. Gabbay and T. Maibaum, Eds. Oxford University Press, *Handb. Log. Comput. Sci.* **1** (1992) 1–116.

[43] N. Martí-Oliet and J. Meseguer, From Petri nets to linear logic through categories: A survey. *Intrenat. J. Foundations Comput. Sci.* **4** (1991) 297–399.

[44] J. Meseguer, Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.* **96** (1992) 73–155.

[45] J. Meseguer and U. Montanari, Petri nets are monoids. *Inform. and Comput.* **88** (1990) 105–155.

[46] H. Miyoshi, Rewriting logic for cyclic sharing structures, T. Sato and A. Middeldorp, Eds., *Fuji International Symposium on Functional and Logic Programming*. World Scientific (1998) 167–186.

[47] U. Montanari and F. Rossi, Contextual nets. *Acta Inform.* **32** (1995) 545–596.

[48] M.J. Plasmeijer and M.C.J.D. van Eekelen, *Functional Programming and Parallel Graph Rewriting*. Addison Wesley (1993).

[49] A.J. Power, An abstract formulation for rewrite systems, D.H. Pitt, D.E. Rydehard, P. Dybjer, A.M. Pitts and A. Poigné, Eds., *Category Theory and Computer Science*. Springer Verlag, *Lecture Notes in Comput. Sci.* **389** (1989) 300–312.

[50] W. Reisig, *Petri Nets: An Introduction*. EACTS Monographs on Theoretical Computer Science. Springer Verlag (1985).

[51] D.E. Rydehard and E.G. Stell, Foundations of equational deductions: A categorical treatment of equational proofs and unification algorithms, D.H. Pitt, A. Poigné and D.E. Rydehard, Eds., *Category Theory and Computer Science*. Springer Verlag, *Lecture Notes in Comput. Sci.* **283** (1987) 114–139.

[52] D. Scott, The lattice of flow diagrams, E. Engeler, Ed., *Semantics of Algorithmic Languages*, Springer Verlag, *Lecture Notes in Math.* **182** (1971) 311–366.

[53] M.R. Sleep, M.J. Plasmeijer and M.C. van Eekelen, *Term Graph Rewriting: Theory and Practice*. Wiley (1993).

[54] J. Staples, Computation of graph-like expressions. *Theoret. Comput. Sci.* **10** (1980) 171–195.

[55] J. Staples, Optimal evaluation of graph-like expressions. *Theoret. Comput. Sci.* **10** (1980) 297–316.

[56] J. Staples, Speeding up subtree replacement systems. *Theoret. Comput. Sci.* **11** (1980) 39–47.

[57] R.H. Street, Categorical structures, M. Hazewinkel, Ed., *Handbook of Algebra.* Elsevier (1996) 529–577.

[58] D.A. Turner, A new implementation technique for applicative languages. *Software: Practice and Experience* **9** (1979) 31–49.

[59] M. Wand, A concrete approach to abstract recursive definitions, M. Nivat, Ed., *Automata, Languages and Programming.* North Holland (1973) 331–341.