Extending Graph Rewriting with Copying

Erik Barendsen Sjaak Smetsers

University of Nijmegen, Computing Science Institute
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
e-mail erikb@cs.kun.nl, sjakie@cs.kun.nl, fax +31.80.652525.

Abstract. The notion of term graph rewrite system (TGRS) is extended with a *lazy copying* mechanism. By analyzing this mechanism, a confluence result is obtained for these so-called *copy term graph rewrite systems* (C-TGRS). Some ideas on the use of lazy copying in practice are presented.

1 Introduction

There are several models of computation that can be viewed as a basis for functional programming. Traditional examples are the *lambda calculus* and *term rewrite systems*. Graph rewriting is a relatively new concept, providing a model that is sufficiently elegant and abstract, but at the same time incorporates mechanisms that are more realistic with respect to actual implementation techniques.

Graph rewrite systems were introduced in Barendregt et al. [1987b]. The present paper deals with a restricted form of GRS's: the so-called *term graph rewrite systems* (TGRS's, see Barendregt et al. [1987a]). An alternative approach to graph rewriting, based on category theory, can be found in Ehrig et al. [1987]. TGRS's are very well suited as a basis for implementation of functional languages. This is demonstrated by the 'intermediate language' CLEAN that is based on TGRS's (see Brus et al. [1987]). Sequential implementations based on graph rewriting turned out to be successful (see Smetsers et al. [1991]). Although pure TGRS's are convenient to model *sequential* computations, they do not capture some aspects that are important in *parallel* implementations.

In a functional program the evaluation result of any expression is independent of the chosen reduction order by the absence of side effects. This makes functional languages attractive for implementation on parallel hardware. An important class of parallel machines is formed by systems of loosely coupled processors, each equipped with its own local memory. An implementation of functional languages on such a machine has to deal with the exchange of data during program execution. This is a nontrivial matter. In the literature, however, the communication mechanism is often considered as a pure implementation issue and therefore handled *ad hoc*.

In the 'overall view' of a computation, the communication between two processors involves duplication of data. In order to investigate theoretical properties of such a computation, graph rewriting systems will be extended with a general copying mechanism. The information transfer is determined by the moment a processor needs data, and by the amount of information that is available (i.e. not subject to a present computation) at that moment. In terms of copying this means that it should be possible to defer copying at some points in the subject graph until these parts of the computation have been finished. This way of copying is called *lazy copying*. Lazy copying can be used to model communication *channels*, as proposed in van Eekelen et al. [1991]. Together with a mechanism for creating parallel reduction processes, the lazy copying formalism provides a specification method for arbitrary process topologies with various kinds of communication links.

In section 5 some approaches to copying are discussed, and the concept of lazy copying is formalized, resulting in the notion of *copy term graph rewrite system* (C-TGRS). Then some properties of C-TGRS are investigated, leading to the proof of a confluence result. Finally the use of lazy copying in practice is considered. Although we consider a restricted class of TGRS's, we expect that our results can be generalized further.

It turns out that the extension of TGRS's with copying can be described in an elegant way. It is also attractive from a theoretical point of view: such extended TGRS's provide an intermediate formalism between TRS's (where inputs of functions are duplicated during a rewrite step $\mathbf{F}(x) \to \mathbf{G}(x,x)$) and TGRS's (where this duplication does not take place).

The notion of TGRS is still very fresh. Due to the lack of standard terminology, notation, and basic theory it was necessary to include a rather extensive section on the foundations of graph rewriting. We aim at an operational description in the style of existing syntactical work in lambda calculus and term rewriting. For more details and examples the reader is referred to Barendsen and Smetsers [1992]. The theory developed there provides an useful framework for the description and the analysis of complex graph operations.

2 Graphs

This section summarizes some basic notions that were presented in Barendsen and Smetsers [1992].

The objects of our interest are directed graphs in which each node is labeled with a specific symbol. The number of outgoing edges of a node is determined by its symbol. In the sequel we assume that $\mathcal{N}$ is some basic set of *nodes* (infinite; one often takes $\mathcal{N} = \mathbb{N}$), and Σ is a (possibly infinite) set of *symbols* with *arity* in $\mathbb{N}$.

Definition 1. (i) A *labeled graph* (over $\langle \mathcal{N}, \Sigma \rangle$) is a triple

$$g = \langle N, symb, args \rangle$$

such that

(a) $N \subseteq \mathcal{N}$; N is the set of *nodes* of g;
(b) $symb : N \to \Sigma$; $symb(n)$ is the *symbol* at node n;
(c) $args : N \to N^*$ such that $\mathrm{length}(args(n)) = \mathrm{arity}(symb(n))$.

Thus $args(n)$ specifies the outgoing edges of n. The i-th component of $args(n)$ is denoted by $args(n)_i$.

(ii) A *rooted graph* is a quadruple

$$g = \langle N, symb, args, r \rangle$$

such that $\langle N, symb, args \rangle$ is a labeled graph, and $r \in N$. The node r is called the *root* of the graph g.

(iii) The collection of all finite rooted labeled graphs over $\langle \mathcal{N}, \Sigma \rangle$ is indicated by $\mathbb{G}$.

Convention. (i) $m, n, n', \ldots$ range over nodes; $g, g', h, \ldots$ range over (rooted) graphs.

(ii) If g is a (rooted) graph, then its components are referred to as N_g, $symb_g$, $args_g$ (and r_g) respectively.

(iii) To simplify notation we usually write $n \in g$ instead of $n \in N_g$.

Definition 2. (i) A *path* in a graph g is a sequence $p = (n_0, i_0, n_1, i_1, \ldots, n_\ell)$ where $n_0, n_1, \ldots, n_\ell \in g$, and $i_0, i_1, \ldots, i_{\ell-1} \in \mathbb{N}$ are 'edge specifications' such that $n_{k+1} = args(n_k)_{i_k}$ for all $k < \ell$. In this case p is said to be a *path from* n_0 *to* n_ℓ (notation $p : n_0 \rightsquigarrow n_\ell$).

(ii) Let p be as above. The *length* of p (notation $|p|$) is ℓ. p *contains* n (notation $n \in p$) if $n_k = n$ for some $k \leq \ell$. p is a *rooted path* if it starts with the root of the graph (i.e. $p : r_g \rightsquigarrow n$ for some $n \in g$). The collection of rooted paths in g is denoted by $\mathrm{RP}(g)$. p is *cyclic* if $n_j = n_k$ for some $j \neq k$.

(iii) Let $m, n \in g$. m is *reachable from* n (notation $n \rightsquigarrow m$) if $p : n \rightsquigarrow m$ for some path p in g.

Definition 3. Let g be a rooted graph.

(i) g is *coherent* if $\forall n \in g \;\; \exists p \;\;\; p : r_g \rightsquigarrow n$.

(ii) g is a *tree* if $\forall n \in g \;\; \exists! p \;\;\; p : r_g \rightsquigarrow n$.

Definition 4. Let g be a graph and $n \in g$. The *subgraph of* g *at* n (notation $g \,|\, n$) is the rooted graph $\langle N, symb, args, n \rangle$ where $N = \{m \in g \mid n \rightsquigarrow m\}$, and $symb$ and $args$ are the restrictions (to N) of $symb_g$ and $args_g$ respectively.

Below we describe the relation $\preceq$, induced by structure preserving mappings called graph homomorphisms.

Definition 5. Let g, h be graphs, and $\varphi : N_g \to N_h$.

(i) φ is a *(graph) homomorphism* from g to h (notation $\varphi : g \to h$) if for all $n \in g$

$$\begin{aligned} symb_h(\varphi(n)) &= symb_g(n), \\ args_h(\varphi(n))_i &= \varphi(args_g(n)_i). \end{aligned}$$

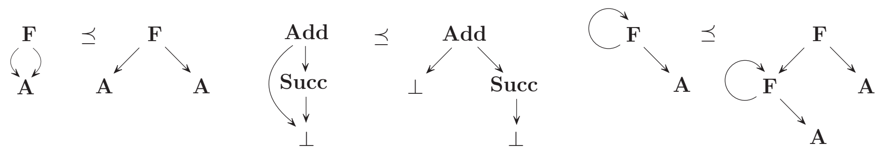
(ii) φ is a *rooted homomorphism* (notation $\varphi : g \xrightarrow[\mathrm{r}]{} h$) if $\varphi : g \to h$ and moreover $\varphi(r_g) = r_h$.

Definition 6. The ordering $\preceq$ on $\mathbb{G}$ is defined as follows.

$$g \preceq h \quad \text{if} \quad \varphi : h \underset{\mathrm{r}}{\to} g \text{ for some } \varphi.$$

Here h can be seen as the result of partially *unraveling* g. Sometimes we write $g \preceq_{\varphi} h$ to indicate the homomorphism explicitly.

Example 1. Our suggestive drawings of graphs will be self-explanatory.



To each graph g one can associate a (possibly infinite) tree $\mathrm{U}(g)$ by an operation called *complete unraveling*. By $g \sim_{\mathrm{T}} g'$ we denote that g and g' are *tree equivalent*, i.e. have the same unraveling.

Definition 7. g is *equivalent* to h (notation $g \equiv h$) if there exists a bijective rooted homomorphism from g to h. We will usually identify equivalent graphs and replace $\equiv$ by $=$.

Definition 8. Let $g, g' \in \mathbb{G}$.
(i) g and g' are *upward compatible* (notation $g \uparrow g'$) if for some $h \in \mathbb{G}$ one has $g \preceq h$ and $g' \preceq h$.
(ii) g and g' are *downward compatible* (notation $g \downarrow g'$) if for some $h \in \mathbb{G}$ one has $h \preceq g$ and $h \preceq g'$.

In Barendsen and Smetsers [1992] it is proved that the set of *regular graphs*

$$\mathbb{G}^* = \{g \in \mathrm{GRAPHS} \mid g' \preceq g \ \text{ for some } g' \in \mathbb{G}\},$$

equipped with the ordering $\preceq$, forms an unpointed cpo. Using the same proof technique one can show the following.

Theorem 9. (i) $g \sim_{\mathrm{T}} g' \;\Rightarrow\; g \uparrow g'$.
(ii) $g \downarrow g' \;\Rightarrow\; g \uparrow g'$.

Proof. See Barendsen and Smetsers [1992]. In fact something stronger is proved: one can construct a $\preceq$-smallest common unraveling. $\square$

This result allows us to reason about compatible graphs entirely within the collection of finite graphs: a translation to the (possibly infinite) complete unravelings is not needed.

3 Graph rewriting

This section briefly describes graph rewriting by means of certain elementary operations.

Definition 10. Let $\perp$ be a special symbol in Σ with arity 0. Let g be a graph.

(i) The set of *empty nodes* of g (notation g°) is the collection

$$g^\circ = \{n \in g \mid symb_g(n) = \perp\}.$$

(ii) The set of *non-empty nodes* (or *interior*) of g is denoted by $g^\bullet$. So $N_g = g^\circ \cup g^\bullet$.

(iii) g is *closed* if $g^\circ = \emptyset$.

The objects on which computations are performed are closed graphs; the others are used as auxiliary objects, e.g. for defining graph rewrite rules.

Definition 11. (i) A *term graph rewrite rule* (or *rule* for short) is a triple $R = \langle g, l, r\rangle$ where g is a (possibly open) graph, and $l, r \in g$ (called the *left root* and *right root* of R), such that

(a) $(g \mid l)^\bullet \neq \emptyset$;

(b) $(g \mid r)^\circ \subseteq (g \mid l)^\circ$.

(ii) If $symb_g(l) = \mathbf{F}$ then R is said to be a *rule for* $\mathbf{F}$.

(iii) R is *left-linear* if $g \mid l$ is a tree.

Here condition (1) expresses that the left-hand side of the rewrite rule should not be just a variable. Moreover condition (2) states that all variables occurring on the right-hand side of the rule should also occur on the left-hand side.

Notation. We will write $R \mid l$, $R \mid r$ for $g_R \mid l_R$, $g_R \mid r_R$ respectively.

Definition 12. Let p, g be graphs.

(i) A *matching homomorphism* (or *match* for short) from p to g is a function $\mu : N_p \to N_g$ such that for all $n \in p^\bullet$

$$\begin{aligned} symb_g(\mu(n)) &= symb_p(n), \\ args_g(\mu(n))_i &= \mu(args_p(n)_i). \end{aligned}$$

In this case we write $\mu : p \underset{\mathrm{m}}{\rightarrow} g$.

(ii) μ is a *rooted match* from p to g (notation $\mu : p \underset{\mathrm{rm}}{\rightarrow} g$) if $\mu : p \underset{\mathrm{m}}{\rightarrow} g$ and $\mu(r_p) = \mu(r_g)$.

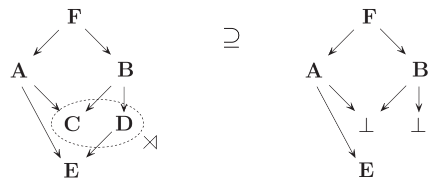
Definition 13. Let $s, g \in \mathbb{G}$. Then s is a *segment* of g (notation $s \subseteq g$) if s is coherent, and s results (up to isomorphism) from g by replacing some subgraphs by $\perp$. More formally, $s \subseteq g$ if there exists an injective match $\iota : s \underset{\mathrm{rm}}{\rightarrow} g$. Observe that such an injection is unique; it will be referred to as $\iota_{s,g}$, or just ι if there is no danger of confusion.

Intuitively, $s \subseteq g$ if s is an initial ('rooted') part of g. The selection of a segment of a given graph is indicated as follows. The construction itself is left implicit.

Definition 14. Let g be a graph and $S \subseteq N_g$. The *segment* of g *up to* S (notation $g \rtimes S$) is obtained by replacing the S-nodes in g by $\perp$.

A segment is obtained by cutting off subgraphs: the symbol $\rtimes$ is meant to resemble a pair of scissors.

Example 2.



Traditionally, a redirection in a graph g is an operation which replaces all references to a given node by references to another one. This notion is generalized in the following definition.

Definition 15. (i) A *redirection map* (or simply *redirection*) on g is a function $\rho : N_g \to N_g$.
(ii) ρ is *projective* if $\rho \circ \rho = \rho$.
(iii) Let $N \subseteq N_g$. N is *ρ-closed* if $\rho(N) \subseteq N$.
(iv) Let h be a (sub)graph. Then ρ is *applicable* to h if N_h is ρ-closed.
(v) The *field* of ρ (notation $\mathrm{Fld}(\rho)$) is the set

$$\mathrm{Fld}(\rho) = \{n \in N_g \mid \rho(n) \neq n\}.$$

(vi) ρ is a *single redirection* if $\mathrm{Fld}(\rho)$ is a singleton. In this case ρ is written as $[m \mapsto n]$ if $\mathrm{Fld}(\rho) = \{m\}$ and $\rho(m) = n$.

A redirection map on g induces a transformation of g, as indicated before.

Definition 16. (i) Let ρ be a redirection on g. The result of *applying* ρ to g (notation $\rho(g)$) is the graph

$$\rho(g) = \langle N_g, symb_g, \overline{\rho} \circ args_g, \rho(r_g) \rangle$$

where $\overline{\rho}$ is the extension of ρ to sequences, i.e. $\overline{\rho}(n_1, \ldots, n_k) = (\rho(n_1), \ldots, \rho(n_k))$.
(ii) The application $[m \mapsto n](g)$ is traditionally denoted as $g[m := n]$. (This notation will also occasionally be used in the degenerated case $m = n$.)

Below a method for extending graphs will be described. In order to allow broader applications the description is more general than is needed for pure graph rewriting.

Definition 17. Let N, A be sets of nodes. The *disjoint set extension* of N with A (notation $N \cup^+ A$) is the set $N \cup A^*$, where $A^* = \{a^* \mid a \in A\}$ is a set of fresh nodes associated with A.

Convention. There is a canonical way of extending an operation f on N or A to an operation f^* on $N \cup^+ A$. If no confusion can arise, we keep writing f, a for f^*, a^* respectively. The same will be done for subsets of N, A and the corresponding subsets of $N \cup^+ A$.

Definition 18. Let g be a graph.

(i) Let h be a graph, $H \subseteq N_h$, and $\mu : (N_h - H) \to N_g$. The *extension* of g with $\langle h, H \rangle$ according to μ (notation $g +_\mu \langle h, H \rangle$) is defined as follows. Set $g +_\mu \langle h, H \rangle = k$, where

$$\begin{aligned}
N_k &= N_g \cup^+ H; \\
\mathit{symb}_k(n) &= \mathit{symb}_g(n) && \text{if } n \in g, \\
&= \mathit{symb}_h(n) && \text{otherwise;} \\
\mathit{args}_k(n)_i &= \mathit{args}_g(n)_i && \text{if } n \in g, \\
&= \mathit{args}_h(n)_i && \text{if } n, \mathit{args}_h(n) \in H, \\
&= \mu(\mathit{args}_h(n)_i) && \text{otherwise;} \\
r_k &= r_g.
\end{aligned}$$

(ii) An *extension specification* (or *extension* for short) for g is a triple $E = \langle h, H, \mu \rangle$ with h, H and μ as above. Then the extension of g with E is

$$g + E = g +_\mu \langle h, H \rangle.$$

(iii) The *extension root* of E (notation $\mathrm{r}(E)$) is defined as follows.

$$\begin{aligned}
\mathrm{r}(E) &= r_h && \text{if } r_h \in H, \\
&= \mu(r_h) && \text{otherwise.}
\end{aligned}$$

Notation. We use $\langle h, \mu \rangle$ as an abbreviation for $\langle h, h^\bullet, \mu \rangle$. Analogously we write $g +_\mu h$ for $g +_\mu \langle h, h^\bullet \rangle$.

Definition 19. Let g be a rooted graph. The result of performing *garbage collection* on g (notation $\mathrm{GC}(g)$) is the graph obtained from g by deleting all nodes that are not reachable from its root, i.e. $\mathrm{GC}(g) = g \mid r_g$.

The three basic operations extension, redirection, and garbage collection are combined into a single operation called *replacement*. This mechanism corresponds to *substitution* in lambda calculus and term rewriting.

Definition 20. Let E be an extension of g, and let $n \in g$. The result of *replacing* n by E in g (notation $g[n := E]$) is defined by

$$g[n := E] = \mathrm{GC}((g + E)[n := \mathrm{r}(E)]).$$

Definition 21. Let g be a graph, and $\mathcal{R}$ a set of rewrite rules.

(i) An $\mathcal{R}$*-redex* in g (or just *redex*) is a tuple $\Delta = \langle R, \mu \rangle$ where $R \in \mathcal{R}$, and $\mu : (R \mid l) \underset{\mathrm{m}}{\rightarrow} g$. The node $\mu(l)$ is called the *redex root* of Δ.

(ii) The extension associated with Δ is

$$E_\Delta = \langle R \mid r,\ (R \mid r)^\bullet - (R \mid l)^\bullet,\ \mu \rangle.$$

We write $g + \Delta$ instead of $g + E_\Delta$, and $g[n := \Delta]$ instead of $g[n := E_\Delta]$. Moreover $\mathrm{r}(\Delta)$ means $\mathrm{r}(E_\Delta)$.

(iii) Let $\Delta = \langle R, \mu \rangle$ be a redex. The result g' of *contracting* Δ in g is given by the replacement

$$g' = g[\mu(l) := \Delta].$$

In this case we say that g reduces to g' via Δ (notation $g \overset{\Delta}{\underset{\mathcal{R}}{\rightarrow}} g'$, or just $g \underset{\mathcal{R}}{\rightarrow} g'$).

The following results concern the interaction between the basic graph operations.

Lemma 22. *Let g be a graph, $E = \langle h, I, \mu \rangle$ an extension of g, and let ρ be a redirection.*

(i) $\mathrm{GC}(\rho(\mathrm{GC}(g))) = \mathrm{GC}(\rho(g))$.

(ii) $\mathrm{GC}(\rho(\mathrm{GC}(g) + E)) = \mathrm{GC}(\rho(g) + E)$.

Proposition 23. *Let $\langle h, H, \mu \rangle$ be an extension of g, and let ρ_1, ρ_2 be redirections of g, h respectively such that $N_h - H$ is ρ_2-closed. Then*

$$\rho_1 \circ \mu \circ \rho_2(g +_{\mu \circ \rho_2} \langle h, H \rangle) = \rho_1(g) +_{\rho_1 \circ \mu} \langle \rho_2(h), H \rangle.$$

(Here μ is tacitly extended to all nodes of the extended graph, by defining $\mu(n) = n$ if $n \notin N_h - H$. The same is done for ρ_1 and ρ_2.)

4 Graph rewrite systems

A collection of graphs and a set of rewrite rules can be combined into a graph rewrite system. A special class of so-called orthogonal graph rewrite systems is the subject of further investigations.

Definition 24. A *term graph rewrite system* (TGRS) is a tuple $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{R}$ is a set of rewrite rules, and $\mathcal{G} \subseteq \mathbb{G}$ is a set of closed graphs which is closed under $\mathcal{R}$-reduction.

It appears that under certain circumstances the order of reduction can be changed without influencing the final result. This is made more explicit below.

Definition 25. Let $\Delta_1 = \langle R_1, \mu_1 \rangle$ and $\Delta_2 = \langle R_2, \mu_2 \rangle$ be redexes in g.

(i) Δ_1 and Δ_2 are *disjoint* if $\mu_1(l_1) \notin \mu_2(R_2 \mid l_2)^\bullet$ and $\mu_2(l_2) \notin \mu_1(R_1 \mid l_1)^\bullet$.

(ii) Δ_1 and Δ_2 *interfere* if $\mu_1(l_1) = \mathrm{r}(\Delta_2)$ and $\mu_2(l_2) = \mathrm{r}(\Delta_1)$.

Definition 26. Let $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS.

(i) $\mathcal{T}$ is *left-linear* if each $R \in \mathcal{R}$ is left-linear.

(ii) $\mathcal{T}$ is *regular* if for each $g \in \mathcal{G}$ the $\mathcal{R}$-redexes in g are pairwise disjoint.

(iii) $\mathcal{T}$ is *orthogonal* if $\mathcal{T}$ is both left-linear and regular.

(iv) $\mathcal{T}$ is *interference-free* if $\mathcal{G}$ does not contain any graph with interfering redexes.

Theorem 27. *Let $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be orthogonal and interference-free. Then $\mathcal{R}$-reduction satifies the Church-Rosser property. In a picture:*

$$\begin{tikzcd}
 & \cdot \arrow[dl, two heads, "\mathcal{R}"'] \arrow[dr, two heads, "\mathcal{R}"] & \\
\cdot \arrow[dr, two heads, dotted, "\mathcal{R}"] & & \cdot \arrow[dl, two heads, dotted, "\mathcal{R}"'] \\
 & \cdot &
\end{tikzcd}$$

Remark 28. If an orthogonal TGRS $\mathcal{T}$ is cycle-free, i.e. the graphs of $\mathcal{T}$ do not contain cycles, then $\mathcal{T}$ is interference-free.

Unraveling a graph might influence the redexes of that graph. The following result implies that unraveling and ordinary reduction can be combined preserving the Church-Rosser property (see Barendsen and Smetsers [1992]).

Proposition 29. *Let g be cycle-free. Suppose $g \preceq g'$ and $g \underset{\mathcal{R}}{\rightarrow} h$. Then $g' \underset{\mathcal{R}}{\twoheadrightarrow} h'$ for some $h' \succeq h$.*

In order to decribe the combination of unraveling and ordinary reduction we denote unraveling as a rewrite step, i.e. $g \underset{\preceq}{\rightarrow} g'$ if $g \preceq g'$.

Theorem 30. *Let $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be orthogonal and cycle-free. Then $\mathcal{R}\preceq$-reduction is Church-Rosser.*

$$\begin{tikzcd}
 & \cdot \arrow[dl, two heads, "\mathcal{R}\preceq"'] \arrow[dr, two heads, "\mathcal{R}\preceq"] & \\
\cdot \arrow[dr, two heads, dotted, "\mathcal{R}\preceq"'] & & \cdot \arrow[dl, two heads, dotted, "\mathcal{R}\preceq"] \\
 & \cdot &
\end{tikzcd}$$

5 Copying

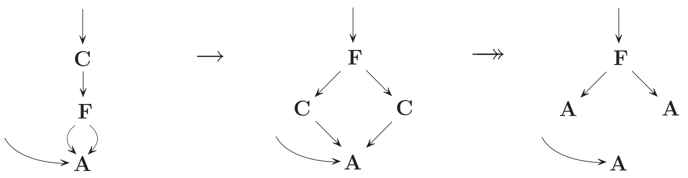
Extending graph rewriting with a copying mechanism can be done in several ways. Below some of these are discussed.

Intuitively, copying a graph causes a new graph to be created that has the same structure as, but no nodes in common with the original graph. Let **C** be a unary symbol in Σ. A copy node in a graph g is a node with symbol **C**. The intended meaning is that evaluation of a copy node leads to duplication of the graph it refers to. We wish to describe how such a **C**-node can be handled.

The first possibility is to extend an existing TGRS $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ to a system $\mathcal{T}^{\mathrm{C}}$ by adding new rewrite rules for copy reduction to $\mathcal{R}$, thus obtaining $\mathcal{R}^{\mathrm{C}}$. This can be done in various ways, the simplest of which is adding a rule

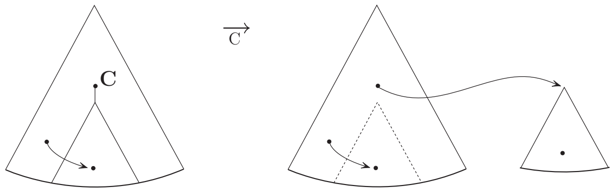
$$\mathbf{C}(\mathbf{F}(x_1, \ldots, x_n)) \to \mathbf{F}(\mathbf{C}(x_1), \ldots, \mathbf{C}(x_n))$$

for each symbol **F** with arity n in $\mathcal{G}$. Although this extension can be described easily, there are several disadvantages. Firstly, the number of extra rewrite rules may be infinite if in $\mathcal{G}$ infinitely many symbols are used. Secondly, mixing copy rules with the reduction rules in $\mathcal{R}$ may destroy properties of $\mathcal{R}$-reduction such as confluence (see section 7), or at least make it very difficult to check whether known properties of $\mathcal{T}$ extend to $\mathcal{T}^{\mathrm{C}}$. This is reinforced by the fact that copying is done stepwise and therefore cannot be considered as a basic action such as redirection or extension. Another unwanted effect is that sharing cannot be maintained, as is illustrated below. The loose arrows indicate edges coming in from the surrounding graph.

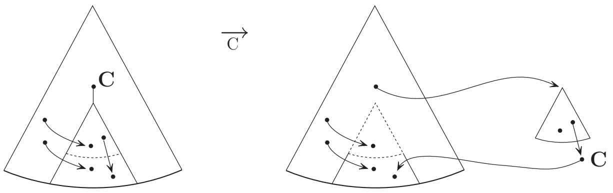


Thus shared redexes below **F** are duplicated. This last drawback could in principle be overcome by using more sophisticated rewrite rules.

The second possibility is the addition of a new copy mechanism to graph rewriting. A rude solution involves replacing the **C**-node by a copy of the whole subgraph it refers to, resulting in so-called *eager copying*. It is visualized in the following figure.



Thus sharing below the copy node remains intact. However, all redexes below that node are duplicated, involving 'duplication of work'. It may therefore be convenient to duplicate only some redexes of the subgraph in question and postpone the rest of the copy action. This is called *lazy copying*. New copy nodes are added where the copy procedure has been interrupted. In a picture this looks as follows.



Of course, this mechanism is more complicated to describe than the previous one. Moreover, if one uses a reduction strategy to select a specific reduction order matters become complicated because one has to decide at which point to interrupt the copying. We will go into this in section 8.

The rest of this section is devoted to a formal description of lazy copying.

Definition 31. Let g be a graph.

(i) Let $n \in g$ and $s \subseteq g \mid n$. Note that $\langle s, \iota_{s,g} \rangle$ is an extension of g. The *s-expansion of g at n* (notation $g[n := s]$) is defined by

$$g[n := s] = g[n := \langle s, \iota_{s,g} \rangle].$$

(ii) g' is an *expansion* of g if $g' = g[n := s]$ for some n and s.

Lemma 32. *Let $s \subseteq g \mid n$. Then $g \preceq g[n := s]$.*

Definition 33. Let $\mathbf{C}$ be a special symbol in Σ with arity 1.

(i) A node $n \in g$ is a *copy node* if $symb_g(n) = \mathbf{C}$. The set of copy nodes of g is denoted by $\mathrm{C}(g)$.

(ii) The definition of *term graph rewrite rule* (11) is extended with

(3) $\mathrm{C}(g \mid l) = \emptyset$, if $R = \langle g, l, r \rangle$ is a rule.

This requirement states that only the right-hand side of a rewrite rule may contain copy nodes. Thus copy nodes may not be used as part of a pattern.

Definition 34. Let g be a graph, and $M \subseteq N_g$. The *copy indirection* of M in g (notation $\mathrm{CI}(g, M)$) is the graph that is obtained from g by replacing each node $n \in M$ to a new copy node with n as argument.

Notation. If n is a unary node in g then $args_g(n)_1$ is written as $succ(n)$.

Definition 35. (i) A *copy redex* in a graph g is a pair $\langle n, s \rangle$ where

(a) $n \in \mathrm{C}(g)$

(b) $s \subseteq g \mid succ(n)$ and $s^\bullet \neq \emptyset$.

(ii) Let $\Delta = \langle n, s \rangle$ be a copy redex in g. The result g' of *contracting* Δ in g is defined by the replacement

$$g' = g[n := \mathrm{CI}(s, s^\circ)],$$

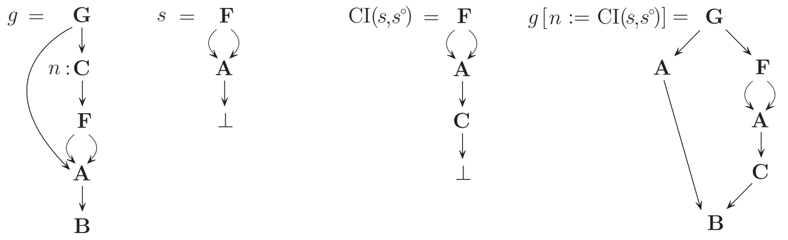
which is shorthand for $g[n := \langle \mathrm{CI}(s, s^\circ), \iota \rangle]$.

(iii) We write $g \overset{\Delta}{\underset{C}{\rightarrow}} g'$ to indicate that g *copy-reduces* to g' via redex Δ.

Definition 36. A *copy term graph rewrite system* (C-TGRS) is a structure $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{G} \subseteq \mathbb{G}$ is a set of graphs, closed under $\mathcal{R}$-reduction and C-reduction.

Remark 37. Notions introduced for TGRS's (such as left-linearity and orthogonality) are used for C-TGRS's as well by considering them as ordinary TGRS's.

Example 3. For the indicated graph g and segment $s \subseteq g$, the copy indirection of s and the result of contracting copy redex $\langle n, s \rangle$ in g is displayed below.



6 Analyzing copy reduction

Because lazy copying is a complicated concept, we will decompose copy reduction into three operations, namely
(1) partial unraveling (expansion),
(2) addition of **C**-nodes,
(3) elimination of **C**-nodes.
The main part of this section addresses the **C**-elimination mechanism and its interaction with the three basic graph operations mentioned in section 3, and with unraveling. Then it is shown that copy reduction can indeed be splitted in the above-mentioned way.

The elimination of **C**-nodes can be viewed as a redirection. This is made precise in the following.

Definition 38. Let g be a rooted graph, and D be a collection of of nodes of g. A path p in g is *within* D (notation $p \subseteq D$) if $n \in D$ for all $n \in p$. D is *cycle-free* (*in* g) if each path of g within D are a-cyclic.

Remark 39. Let g be graph. Let $D \subseteq g$ be a finite set of unary nodes. Then every a-cyclic path p in D is uniquely determined by its begin and end points, for each node of p has exactly one successor. Consequently, if D is cycle-free then for any $n \in g$ there exists exactly one $p : n \rightsquigarrow m$ such that

$$\begin{gathered} m \notin D, \\ \forall m' \in p \quad [m' \neq m \;\Rightarrow\; m' \in D]. \end{gathered}$$

The latter expression is abbreviated to $p - m \subseteq D$.

In the sequel we will only consider finite sets D consisting of nodes with arity 1. Remark 39 enables us to define the following.

Definition 40. Let g be a graph, and $D \subseteq \mathrm{C}(g)$ cycle-free. Define the redirection $\mathrm{e}_{D,g}$ as follows. Let $n \in g$. Then $\mathrm{e}_{D,g}(n)$ is the uniquely determined $m \notin D$ such that $p : n \rightsquigarrow m$ for some p with $p - m \subseteq D$. Then $\mathrm{e}_{D,g}$ is obviously projective, and $\mathrm{Fld}(\mathrm{e}_{D,g}) = D$.

Now the actual elimination of **C**-nodes in g is obtained by applying the corresponding redirection operation to g.

Definition 41. Let g be a graph and $D \subseteq \mathrm{C}(g)$. $\mathrm{E}_D(g)$ denotes the graph that is obtained by deleting all copy nodes of D from g, i.e. $\mathrm{E}_D(g) = \mathrm{e}_{D,g}(g)$.

Proposition 42. *Let g be a graph, $D_1, D_2 \subseteq \mathrm{C}(g)$ such that $D_1 \cup D_2$ is cycle-free.*
(i) $\mathrm{e}_{D_1 \cup D_2, g} = \mathrm{e}_{D_2, \mathrm{E}_{D_1}(g)} \circ \mathrm{e}_{D_1, g}$.
(ii) $\mathrm{E}_{D_1 \cup D_2}(g) = \mathrm{E}_{D_2}(\mathrm{E}_{D_1}(g))$.

The interaction between **C**-elimination and reduction and unraveling respectively can now be described. We spell out a proof in detail to illustrate the usage of the properties of the basic operations.

Proposition 43. *Let g be a graph, $m, n \in g$, and let $D \subseteq g$ be cycle-free in both g and $g[n := m]$ such that $n \notin D$.*
(i) $[n \mapsto \mathrm{e}_{D,g}(m)] \circ \mathrm{e}_{D,g} = \mathrm{e}_{D,g[n:=m]} \circ [n \mapsto m]$.
(ii) $\mathrm{E}_D(g[n := m]) = \mathrm{E}_D(g)[n := \mathrm{e}_{D,g}(m)]$.

Proposition 44. *Let $g \underset{\mathcal{R}}{\rightarrow} h$. Let $D \subseteq \mathrm{C}(h)$ be cycle-free. Then*

$$\mathrm{GC}(\mathrm{E}_D(g)) \underset{\mathcal{R}}{\rightarrow} \mathrm{GC}(\mathrm{E}_D(h)).$$

Proof. It will be shown that

$$\mathrm{GC}(\mathrm{e}_{D,g}(g)) \underset{\mathcal{R}}{\rightarrow} \mathrm{GC}(\mathrm{e}_{D,h}(h)).$$

Say $g \overset{\Delta}{\underset{\mathcal{R}}{\rightarrow}} h$ with $\Delta = \langle R, \mu \rangle$. Then

$$h = \mathrm{GC}((g + \Delta)[\mu(l) := \mathrm{r}(\Delta)]).$$

Observe that $\mu(R \,|\, l)^\bullet \cap D = \emptyset$ since $(R|l)^\bullet$ is **C**-free. So $\mathrm{e}_{D,g}(n) = n$ for any $n \in \mu(R \,|\, l)^\bullet$. It follows that $\mathrm{e}_{D,g} \circ \mu : (R|l)^\bullet \underset{\mathrm{m}}{\rightarrow} \mathrm{e}_{D,g}(g)$. Write $\mathrm{e}_{D,g}(\Delta) = \langle R, \mathrm{e}_{D,g} \circ \mu \rangle$. Then $\mathrm{e}_{D,g}(\Delta)$ is a redex in $\mathrm{e}_{D,g}(g)$. Note that $\mathrm{r}(\mathrm{e}_{D,g}(\Delta)) = \mathrm{e}_{D,g}(\mathrm{r}(\Delta))$, and $\mathrm{e}_{D,g}(\mu(l)) = \mu(l)$. Contracting $\mathrm{e}_{D,g}(\Delta)$ in $\mathrm{GC}(\mathrm{e}_{D,G}(g))$ results in

$\mathrm{GC}(\ (\ \mathrm{GC}(\mathrm{e}_{D,g}(g)) + \mathrm{e}_{D,g}(\Delta)\ )[\mu(l) := \mathrm{r}(\mathrm{e}_{D,g}(\Delta))]\ )$
$= \mathrm{GC}(\ (\ \mathrm{e}_{D,g}(g) + \mathrm{e}_{D,g}(\Delta)\ )[\mu(l) := \mathrm{r}(\mathrm{e}_{D,g}(\Delta))]\ )$, by lemma 22 (ii)
$= \mathrm{GC}(\ (\ \mathrm{e}_{D,g}(g + \Delta)\ )[\mu(l) := \mathrm{r}(\mathrm{e}_{D,g}(\Delta))]\ )$, by proposition 23

$= \mathrm{GC}(\ (\ \mathrm{e}_{D,g}(g + \mathit{\Delta})\ )[\mu(l) := \mathrm{e}_{D,g}(\mathrm{r}(\mathit{\Delta}))]\ )$
$= \mathrm{GC}(\ (\ \mathrm{e}_{D,\mathrm{GC}(g+\mathit{\Delta})}(g + \mathit{\Delta})\ )[\mu(l) := \mathrm{e}_{D,g}(\mathrm{r}(\mathit{\Delta}))]\ ),$
$= \mathrm{GC}(\ \mathrm{e}_{D,\mathrm{GC}(g+\mathit{\Delta})[\mu(l):=\mathrm{r}(\mathit{\Delta})]}(\ (g + \mathit{\Delta})[\mu(l) := \mathrm{r}(\mathit{\Delta})]\ )\ ),$ by proposition 43 (i)
$= \mathrm{GC}(\ \mathrm{e}_{D,\mathrm{GC}(g+\mathit{\Delta})[\mu(l):=\mathrm{r}(\mathit{\Delta})]}(\ \mathrm{GC}(\ (g + \mathit{\Delta})[\mu(l) := \mathrm{r}(\mathit{\Delta})]\ )\ )\ ),$ by lemma 22 (i)
$= \mathrm{GC}(\ \mathrm{e}_{D,h}(h)\ ).$ □

Proposition 45. *Let* $g \preceq_\varphi h$. *Let* $D \subseteq \mathrm{C}(h)$ *be cycle-free.*
(i) $\mathrm{e}_{\varphi^{-1}(D),g}(g) \preceq_\varphi \mathrm{e}_{D,h}(h)$.
(ii) $\mathrm{GC}(\mathrm{E}_{\varphi^{-1}(D)}(g)) \preceq_\varphi \mathrm{GC}(\mathrm{E}_D(h))$.

Some technical work has to be done in order to split copy reduction into more basic actions.

Lemma 46. *Let* g *be a graph,* $n \in g$*, and* $s \subseteq g \mid n$
(i) *If* n *is unary then* $\mathrm{GC}(\mathrm{E}_{\{r_s\}}(g[n := s])) = \mathrm{GC}(g[n := s \mid \mathit{succ}(n)])$.
(ii) *If* $s^\bullet \neq \emptyset$ *then* $\mathrm{GC}(\mathrm{E}_{(s^\circ)^*}(g[n := \mathrm{CI}(s, s^\circ)])) = \mathrm{GC}(g[n := s])$.

For convenience some new reduction relations are introduced. Remember $g \underset{\preceq}{\rightarrow} h$ iff $g \preceq h$.

Definition 47. The reduction relations $\underset{-}{\rightarrow}$ and $\underset{+}{\rightarrow}$ are defined by

$$g \underset{-}{\rightarrow} h \ \Leftrightarrow\ h = \mathrm{GC}(\mathrm{E}_D(g)) \quad \text{for some nonempty } D \subseteq \mathrm{C}(g),$$
$$g \underset{+}{\rightarrow} h \ \Leftrightarrow\ \mathrm{C}(g) \neq \emptyset \text{ and } g = \mathrm{GC}(\mathrm{E}_D(h)) \quad \text{for some nonempty } D \subseteq \mathrm{C}(h).$$

Proposition 48. $g \underset{\mathrm{C}}{\rightarrow} g' \ \Rightarrow\ g \underset{\preceq}{\rightarrow} \cdot \underset{+}{\rightarrow} \cdot \underset{-}{\rightarrow} g'$.

Proof. Let $\langle n, s\rangle$ be the copy redex of g that is reduced in order to obtain g'. Hence $s \subseteq g \mid \mathit{succ}(n)$ and $g' = \mathrm{GC}(g +_\iota \mathrm{CI}(s, s^\circ)[n := r_s])$. Let $s_\mathrm{c} \subseteq g \mid n$ such that $s = s_\mathrm{c} \mid \mathit{succ}(n)$. Then

$$\begin{aligned} g &\underset{\preceq}{\rightarrow} \mathrm{GC}(g[n := s_\mathrm{c}]), && \text{by lemma 32} \\ &\underset{+}{\rightarrow} \mathrm{GC}(g[n := \mathrm{CI}(s_\mathrm{c}, s_\mathrm{c}^\circ)]), && \text{by lemma 46 (ii)} \\ &\underset{-}{\rightarrow} \mathrm{GC}(g[n := \mathrm{CI}(s, s^\circ)]), && \text{by lemma 46 (i)} \\ &= g'. \ \square \end{aligned}$$

7 Confluence

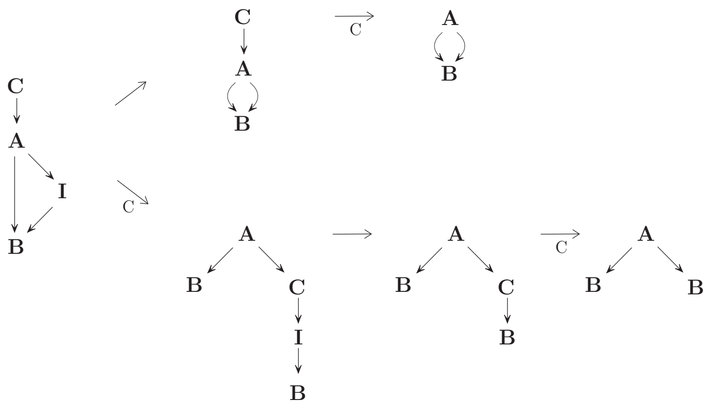
This section starts with the introduction of some standard terminology.

Definition 49. (i) A *reduction relation* on $\mathcal{G}$ is just a binary relation $\boldsymbol{R}$ on $\mathcal{G}$.
(ii) If $\boldsymbol{R}_1, \boldsymbol{R}_2$ are reduction relations, then $\boldsymbol{R}_1\boldsymbol{R}_2$ is $\boldsymbol{R}_1 \cup \boldsymbol{R}_2$.
(iii) A graph g $\boldsymbol{R}$*-reduces in one step* to g' (notation $g \underset{R}{\rightarrow} g'$) if $(g, g') \in \boldsymbol{R}$.
(iv) The relations $\underset{R}{\twoheadrightarrow}$, $\underset{R}{\rightarrow}^+$ and $\underset{R}{\rightarrow}^\equiv$ denote the reflexive transitive closure, the transitive closure and the reflexive closure of $\underset{R}{\rightarrow}$ respectively.

We assume the reader is familiar with the notions *normal form*, *reduction path*, etcetera.

Let $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ a orthogonal cycle-free C-TGRS. We want to obtain a Church-Rosser-like result for $\mathcal{T}$. A pure Church-Rosser result for $\mathcal{R}$C-reduction is impossible, as is shown in the following example.

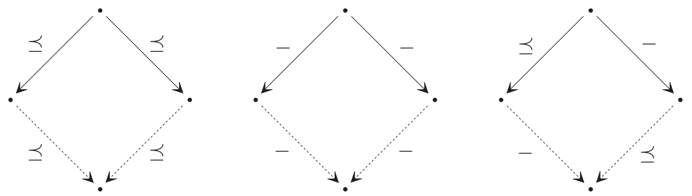
Example 4. Let $\mathbf{I}(x) \to x$ be the usual rule for the identity function. Then one has two 'divergent' reduction paths.



For this reason we study confluence of the system in which the copy action is decomposed. Our aim is to show that $\mathcal{R}\preceq+-$-reduction is confluent.

A useful tool to prove the confluence of a combined reduction relation is the so-called Hindley-Rosen lemma, due to Hindley [1964] and Rosen [1973].

Lemma 50. (i) (ii) (iii)

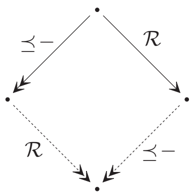


Proof. (i) By theorem 9 (ii).
(ii) By proposition 42 (ii).
(iii) By proposition 45 (ii). □

Proposition 51. $\preceq-$ *is Church-Rosser.*

Proof. By the Hindley-Rosen lemma, using the facts that $\underset{\preceq}{\rightarrow}$ and $\underset{-}{\rightarrow}$ are Church-Rosser (lemma 50 (i, ii)), and that $\underset{\preceq}{\rightarrow}$ and $\underset{-}{\rightarrow}$ commute (lemma 50 (iii)). □

Lemma 52.



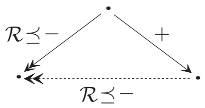
Proof. By propositions 29 and 44 and a diagram chase. □

Theorem 53. *$\mathcal{R}\preceq-$ is Church-Rosser.*

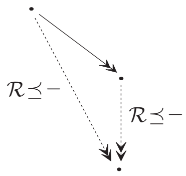
Proof. Again by the Hindley-Rosen lemma, using proposition 51, lemma 52 and theorem 27. □

The confluence proof is carried out in two main steps. First it is shown how +-reduction can be dealth with, and next the Church-Rosser result for $\mathcal{R}\preceq-$-reduction is used. Below, $\to$ stands for $\mathcal{R}\preceq+-$-reduction.

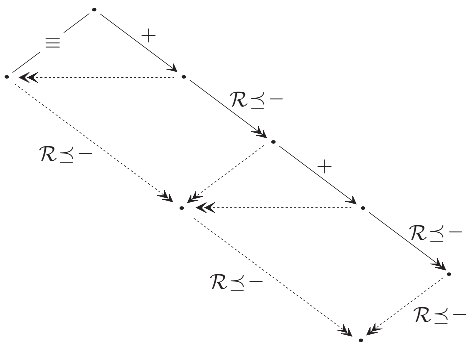
Neutralization Lemma 54.



Proposition 55.



Proof. Using the neutralization lemma (triangles) and theorem 53 (quadrangles) one can erect the following diagram.



□

Theorem 56.

$$\begin{tikzcd}
& \cdot \arrow[dl, two heads] \arrow[dr, two heads] & \\
\cdot \arrow[dr, dotted, two heads, "\mathcal{R}\preceq-"'] & & \cdot \arrow[dl, dotted, two heads, "\mathcal{R}\preceq-"] \\
& \cdot &
\end{tikzcd}$$

Proof. By proposition 55 and theorem 53.

$$\begin{tikzcd}
& \cdot \arrow[dl, two heads] \arrow[dr, two heads] \arrow[ddl, dotted, two heads, "\mathcal{R}\preceq-"] \arrow[ddr, dotted, two heads, "\mathcal{R}\preceq-"'] & \\
\cdot \arrow[d, dotted, two heads, "\mathcal{R}\preceq-"'] & & \cdot \arrow[d, dotted, two heads, "\mathcal{R}\preceq-"] \\
\cdot \arrow[dr, dotted, two heads, "\mathcal{R}\preceq-"'] & & \cdot \arrow[dl, dotted, two heads, "\mathcal{R}\preceq-"] \\
& \cdot &
\end{tikzcd}$$

□

Corollary 57.

$$\begin{tikzcd}
& \cdot \arrow[dl, two heads, "\mathcal{R}\mathrm{C}"'] \arrow[dr, two heads, "\mathcal{R}\mathrm{C}"] & \\
\cdot \arrow[dr, dotted, two heads, "\mathcal{R}\preceq-"'] & & \cdot \arrow[dl, dotted, two heads, "\mathcal{R}\preceq-"] \\
& \cdot &
\end{tikzcd}$$

The relations between various normal forms can now be expressed.

Definition 58. Let g, g' be graphs.

(i) g and g' are *copy tree equivalent* (notation $g \sim_{\mathrm{CT}} g'$) if they are tree equivalent after elimination of all present copy nodes, i.e.

$$\mathrm{E}_{\mathrm{C}(g)}(g) \sim_{\mathrm{T}} \mathrm{E}_{\mathrm{C}(g')}(g').$$

(ii) $\sim_{\preceq-}$ is the reflexive symmetric transitive closure of the reduction relation $\preceq-$.

Theorem 59. *Let $g, g' \in \mathbb{G}$. Suppose $\mathrm{C}(g), \mathrm{C}(g')$ are cycle-free. Then*

$$g \sim_{\mathrm{CT}} g' \;\Leftrightarrow\; g \sim_{\preceq-} g'.$$

Proof. ($\Rightarrow$) Using $\mathrm{U}(\mathrm{GC}(h)) = \mathrm{U}(h)$ one derives

$$\begin{aligned}
g \sim_{\mathrm{CT}} g' &\Rightarrow \mathrm{E}_{\mathrm{C}(g)}(g) \sim_{\mathrm{T}} \mathrm{E}_{\mathrm{C}(g')}(g') \\
&\Rightarrow \mathrm{GC}(\mathrm{E}_{\mathrm{C}(g)}(g)) \sim_{\mathrm{T}} \mathrm{GC}(\mathrm{E}_{\mathrm{C}(g')}(g')) \\
&\Rightarrow \mathrm{GC}(\mathrm{E}_{\mathrm{C}(g)}(g)) \uparrow \mathrm{GC}(\mathrm{E}_{\mathrm{C}(g')}(g')), \quad \text{by theorem 9 (i).}
\end{aligned}$$

($\Leftarrow$) By induction on the generation of $\sim_{\preceq -}$. The crucial case is $g \underset{\preceq}{\to} g'$. Then by lemma 50 (iii) one has $\mathrm{GC}(\mathrm{E}_{\mathrm{C}(g)}(g)) \preceq \mathrm{GC}(\mathrm{E}_{\mathrm{C}(g')}(g'))$ and therefore $\mathrm{E}_{\mathrm{C}(g)}(g) \sim_{\mathrm{T}} \mathrm{E}_{\mathrm{C}(g')}(g')$. $\square$

As a corollary of the Church-Rosser theorem, normal forms turn out to be unique up to copy tree equivalence.

Theorem 60. (i) *If g has g_1, g_2 as $\mathcal{R}$C-normal forms, then $g_1 \uparrow g_2$.*
(ii) *If g has g_1, g_2 as $\mathcal{R}$-normal forms, then $g_1 \sim_{\mathrm{CT}} g_2$.*

8 Towards an implementation of lazy copying

When using term rewriting systems or term graph rewriting systems as a model of computation, these systems are usually equipped with a reduction strategy.

In order to determine *redexes* to be contracted one often uses an auxiliary function selecting *nodes* that are intended to be rewritten in the subject graph. For ordinary orthogonal TGRS's such a node uniquely determines a redex. In the case of C-TGRS's this node can be considered as a partial specification of a redex, since copy redexes consist of a copy node together with an appropriate segment. So a copy node actually specifies a *set* of copy redexes. In this section we will describe a natural method for choosing an appropriate segment, thus obtaining a proper redex selecting procedure.

To have some control over the choice of segments, we extend the language of graph rewriting with the possibility of attaching a so called *defer* attribute to nodes. Intuitively, this attribute temporarily prevents the node in question of being copied. Defer attributes are also allowed to appear in the right-hand side of a rewrite rule. In van Eekelen et al. [1991] these attributes are used to define primitives for specifying communication for various kinds of parallelism.

During graph rewriting, defer attributes are *added* and *propagated* while performing the graph operations extension, redirection and garbage collection, according to the following rules.

- If n is deferred in g, then n is deferred in $g + E$.
- If n is deferred in E, then n^* is deferred in $g + E$.
  (This applies e.g. to $g + \Delta$ for redexes $\Delta = \langle R, \mu \rangle$ in which $R \mid r$ contains deferred nodes.)
- If n is deferred in g, then $\rho(n)$ is deferred in $\rho(g)$.
- If n is deferred in g, and $r_g \rightsquigarrow n$, then n is deferred in $\mathrm{GC}(g)$.

The above-mentioned node selection is made precise in the following. Let $\mathcal{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a C-TGRS with defer attributes.

Definition 61. (i) $\mathrm{D}(g)$ denotes the collection of deferred nodes of g.
(ii) A *pre-strategy* (for $\mathcal{T}$) is a function σ which takes each $g \in \mathcal{G}$ to a set of $\mathcal{R}$-redex roots, and/or copy nodes in g, such that for all $n \in \sigma(g)$

$$n \in \mathrm{C}(g) \;\Rightarrow\; \mathit{succ}(n) \notin \mathrm{D}(g),$$

i.e. σ will not attempt to copy a 'deferred graph'.

(iii) A graph is in σ-*normal form* if $\sigma(g) = \emptyset$.

Now let σ be a pre-strategy for $\mathcal{T}$. Actual rewriting according to σ proceeds as follows. If σ selects an $\mathcal{R}$-redex root, the corresponding redex is contracted. If σ points out a copy node then a copy reduction step is done. The subgraph $g \mid succ(n)$ is copied up to the deferred nodes of g. Thus

$$\begin{aligned} g \underset{\sigma}{\rightarrow} g' \text{ if for some } n \in \sigma(g) \\ & n \in \mathrm{C}(g), \text{ and } g \overset{\langle n,s\rangle}{\underset{\mathrm{C}}{\rightarrow}} g', \\ & \text{where } s = (g \mid succ(n)) \rtimes \mathrm{D}(g), \qquad \text{or} \\ & n \notin \mathrm{C}(g), \text{ and } g \overset{\Delta_n}{\underset{\mathcal{R}}{\rightarrow}} g', \\ & \text{where } \Delta_n \text{ is the } \mathcal{R}\text{-redex associated with } n. \end{aligned}$$

Defer attributes are *removed* during reduction according to the following rule.

- Once a rewrite step is performed, a node $n \in g$ loses its defer attribute if $g \mid n$ is in σ-nf.

As a consequence, each graph in $\mathcal{R}$C-normal form does not have any deferred nodes. Note that the above s is nonempty since $succ(n)$ is not deferred.

The concept of lazy copying, combined with the above segment selection, has been incorporated in the programming language CONCURRENT CLEAN (see Nöcker et al. [1991]). In the implementation of this language on a distributed multi-processor system this technique is used in combination with a parallel version of the so-called *functional* reduction strategy. Communication channels are expressed by using copy nodes. The definition of $\underset{\sigma}{\rightarrow}$ then guarantees that no 'work' is communicated (i.e. no nodes that are marked as 'deferred' are copied from one processor to another). Moreover, the specific definition of the parallel functional reduction strategy guarantees that eventually the necessary information is available by reducing any deferred part of the graph.

The following result formulates the adequacy of reduction in CONCURRENT CLEAN. The details will not be worked out here.

Theorem 62. *In* CONCURRENT CLEAN, *reduction is sound and complete, i.e. normal forms are unique up to tree equivalence, and normal forms can be reached via the above reduction method.*

References

BARENDREGT, H.P., M.C.J.D. VAN EEKELEN, J.R.W. GLAUERT, J.R. KENNAWAY, M.J. PLASMEIJER and M.R. SLEEP

[1987a] Term graph reduction, in: *Proceedings of Parallel Architectures and Languages Europe (PARLE'87), Eindhoven, The Netherlands*, LNCS 259, volume II, Springer-Verlag, Berlin, pp. 141–158.

[1987b] Towards an intermediate language based on graph rewriting, in: *Proceedings of Parallel Architectures and Languages Europe (PARLE'87), Eindhoven, The Netherlands*, LNCS 259, volume II, Springer-Verlag, Berlin, pp. 159–175.

BARENDSEN, E. and J.E.W. SMETSERS
[1992] Graph rewriting and copying, Technical report no. 92-20, Department of Computer Science, University of Nijmegen.

BRUS, T., M.C.J.D. VAN EEKELEN, M.O. VAN LEER and M.J. PLASMEIJER
[1987] Clean: a language for functional graph rewriting, in: *Proceedings of the 3rd International Conference on Functional Programming Languages and Computer Architecture, Portland, Oregon*, LNCS 274, Springer-Verlag, Berlin, pp. 364–384.

VAN EEKELEN, M.C.J.D., M.J. PLASMEIJER and J.E.W. SMETSERS
[1991] Parallel graph rewriting on loosely coupled machine architectures, in: *Proceedings of Conditional and Typed Rewriting Systems (CTRS'90), Montreal, Canada*, LNCS 516, Springer-Verlag, Berlin, pp. 354–369.

EHRIG, H., M. NAGL, G. ROZENBERG and A. ROSENFELD (eds.)
[1987] *Proceedings of the 3rd International Workshop on Graph-Grammars and the Application to Computer Science, Warrenton, Virginia, USA, December 1986*, LNCS 291, Springer-Verlag, Berlin.

HINDLEY, J.R.
[1964] The Church-Rosser property and a result in Combinatory Logic, dissertation, University of Newcastle-upon-Tyne.

NÖCKER, E.G.J.M.H., J.E.W. SMETSERS, M.C.J.D. VAN EEKELEN and M.J. PLASMEIJER
[1991] Concurrent Clean, in: *Proceedings of Parallel Architectures and Languages Europe (PARLE'91), Eindhoven, The Netherlands*, LNCS 505, Springer-Verlag, Berlin, pp. 202–219.

ROSEN, B.K.
[1973] Tree manipulation systems and Church-Rosser theorems, *J. Assoc. Comput. Mach.* 20, pp. 160–187.

SMETSERS, J.E.W., E.G.J.M.H. NÖCKER, J.H.G. VAN GRONINGEN and M.J. PLASMEIJER
[1991] Generating efficient code for lazy functional languages, in: *Proceedings of Conference on Functional Programming Languages and Computer Architecture (FPCA '91), Cambridge, MA*, LNCS 523, Springer-Verlag, Berlin, pp. 592–617.