Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Equational term graph rewriting

Z.M. Ariola and J.W. Klop

Computer Science/Department of Software Technology

CS-R9552 1995

Report CS-R9552
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Equational Term Graph Rewriting

Zena M. Ariola
Computer & Information Science Department
University of Oregon. Eugene, OR 97401, USA
email: ariola@cs.uoregon.edu

Jan Willem Klop
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
Department of Mathematics and Computer Science
Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam
email: jwk@cwi.nl

Abstract

We present an equational framework for term graph rewriting with cycles. The usual notion of homomorphism is phrased in terms of the notion of bisimulation, which is well-known in process algebra and concurrency theory. Specifically, a homomorphism is a functional bisimulation. We prove that the bisimilarity class of a term graph, partially ordered by functional bisimulation, is a complete lattice. It is shown how Equational Logic induces a notion of copying and substitution on term graphs, or systems of recursion equations, and also suggests the introduction of hidden or nameless nodes in a term graph. Hidden nodes can be used only once. The general framework of term graphs with copying is compared with the more restricted copying facilities embodied in the μ-rule, and translations are given between term graphs and μ-expressions. Using these, a proof system is given for μ-expressions that is complete for the semantics given by infinite tree unwinding. Next, orthogonal term graph rewrite systems, also in the presence of copying and hidden nodes, are shown to be confluent.

AMS Subject Classification (1991): 68Q50
CR Subject Classification (1991): F.4.1, F.3.3
Keywords & Phrases: term rewriting systems, term graph rewriting, confluence, bisimulation.
Note: The research of the first author is supported by NSF grant CCR-94-10237. The research of the second author is partially supported by ESPRIT BRA-6454 Confer. Further support for both authors was provided by ESPRIT Working Group 6345 Semagraph. This report is an extended version of the paper with the same title to appear in Acta Informatica.

Table of Contents

1. Introduction

Term rewriting can be divided roughly in first-order term rewriting [DJ90, Klo92], where first-order terms are replaced (reduced, rewritten) according to a fixed set of rewrite rules, and higher-order term rewriting, of which the paradigm is lambda calculus [Bar84]. The distinctive feature of the latter is the presence of bound variables. Term rewriting has become in the last decade a firmly established discipline, with applications in many areas [BE91] such as abstract data types, functional programming, automated theorem proving, and proof theory.

Term graph rewriting originates with the observation that rewrite rules often ask for duplications of subterms. *E.g.,* the Combinatory Logic rewrite rule $\mathsf{S}xyz \longrightarrow xz(yz)$, where the variables x, y, z stand for arbitrary terms, duplicates the term instantiated for z. To save space and time, actual implementations do not perform such a duplication literally, but work instead with pointers to shared subterms. Thus we arrive at rewriting of dags (directed acyclic graphs) rather than terms (finite trees). Recent years have seen the development of the basic theory for acyclic term graph rewriting [BvEG$^+$87, HP88, Plu93, SPvE93]. Typical results are confluence, when an orthogonality restraint is imposed (rules cannot interfere badly with each other) [Sme93], modularity for properties such as confluence and termination (the properties stay preserved in combinations of systems) [Plu93], and adequacy (in what sense is term graph rewriting adequate for ordinary term rewriting) [Ari93, KKSdV94].

Term graph rewriting with cycles goes one step further by admitting cyclic term graphs. These arise naturally when dealing with recursive structures. Classical is the implementation by D. Turner [Tur79] of the fixed point combinator Y by means of a cyclic graph (Figure 1). Only in the last few years a firm foundation of cyclic term graph rewriting has been given, with as a main theorem the confluence property for orthogonal term graph rewriting. Establishing confluence was not altogether trivial since it faced the need for solving the problem of cyclic collapsing terms [Ari92, Ari93, FW91].

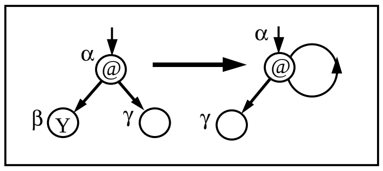


Figure 1.

Previous definitions of term graph rewriting tend to use one of two ways: (1) category-theory oriented [Ken87, Ken88, Ken90, Rao84], (2) implementation-oriented [PvE93]. The first describes graph rewrite steps as push-outs in a category, and some papers have been devoted to analyzing whether this can be done by single or double push-out constructions [Lö93]. The second uses notions like pointers, redirections, indirections. We would like to find an approach that is less 'abstract' than the first, and less 'concrete' than the second.

So, the aim of the present paper is to provide a smooth framework for term graph rewriting, for the general case where cycles are admitted. The starting point is that a cyclic term graph is nothing else than a system of recursion equations. This is an obvious view, however, it seems that the possibilities generated by this point of view have not yet been exploited fully. Some papers indeed describe term graphs as systems of equations, but do not consider the equational transformations that then are possible. As a typical example of the benefits of an

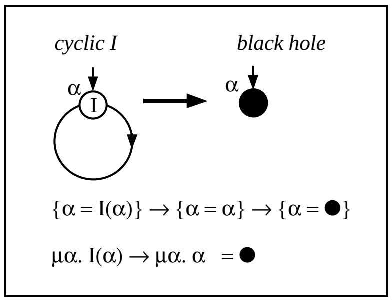


Figure 2.

equational treatment of term graph rewriting, we mention the problem of 'cyclic-I', or more general of cyclic collapsing graphs. See Figure 2. This matter has been called 'a persistent technical problem' [KKSdV94], and indeed several proposals concerning the way that cyclic-I (*i.e.,* the graph $\{\alpha = \mathsf{I}(\alpha)\}$, where I has the 'collapsing' rewrite rule $\mathsf{I}(x) \longrightarrow x$) should be rewritten to, occur in the literature. One possibility is that cyclic-I rewrites to itself. An equational treatment, however, leaves in our opinion no doubt about the proper way of rewriting cyclic-I: it should be rewritten to a new constant that we like to call 'black hole', written as $\bullet$. For graph rewriting, it is as it were a point of singularity; we need a new constant for it to ensure confluence of orthogonal term graph rewriting, even though there is no proper term graph corresponding to it. Interestingly the same observation was made by Corradini in [Cor93] using the cpo approach. This view is also supported by a comparison with the related system of μ-expressions: the new constant $\bullet$ is present there as $\mu\alpha.\alpha$, an expression rewriting only to itself: $\mu\alpha.\alpha \longrightarrow \mu\alpha.\alpha$. In terms of systems of recursion equations, $\bullet$ is the system $\{\alpha = \alpha\}$. The same 'singularity' shows up in the theory of orthogonal infinitary rewriting [KKSdV95]: without more, infinitary confluence fails, but equating all infinite collapsing trees such as I^ω, the infinite unwinding $\mathsf{I}(\mathsf{I}(\mathsf{I}(\mathsf{I}(\mathsf{I}(\cdots)\cdots)$ of either the graph $\{\alpha = \mathsf{I}(\alpha)\}$ or the μ-term $\mu\alpha.\mathsf{I}(\alpha)$, infinitary confluence holds. (It should be mentioned that the 'collapse problem' not only is present with the unary collapse operator I, but also *e.g.,* in the infinitary version or the cyclic graph version of Combinatory Logic with its collapsing rule $\mathsf{K}xy \longrightarrow x$.)

As another example of the ease that Equational Logic introduces for term graph rewriting, we mention: checking bisimilarity of two term graphs in an equational way, somewhat reminiscent to the elegant unification algorithm of Martelli-Montanari [MM82].

An interesting consequence of treating term graphs as systems of recursion equations, is that we are naturally invited to perform the operation of substitution on them. *E.g.,* the graph (or recursion system) $\{\alpha = \mathsf{F}(\beta), \beta = \mathsf{G}(\alpha)\}$, where the first equation always is taken as the leading root equation, can be transformed by substitution to $\{\alpha = \mathsf{F}(\mathsf{G}(\alpha))\}$. Here the node β is 'hidden' or nameless. Thus, allowing the operation of substitution on recursion equations, we get for free a notion of hidden or nameless nodes, a notion that has a certain Linear Logic flavor since it says that some nodes can be used only once, in contrast with ordinary nodes that can be re-used (shared) indefinitely. (But we note that this feature of hidden nodes is not forced upon us; if desired we can avoid it altogether. It is an 'add-on' feature.)

Lambda graph rewriting attempts to do the same as above for lambda calculus. (It is not treated in the present paper.) Acyclic lambda graphs were already considered in the well-known thesis of Wadsworth [Wad71]; and recently there has been a lot of activity concerning them [GAL92], following a solution of Levy's optimality problem for lambda rewriting [Lé80], by Lamping and Kathail [Lam90, Kat90]. As yet, no systematic study has been made of lambda graph rewriting with cycles; but work in progress by the authors [AK94] intends to provide a first step, revealing that there is a remarkable contrast with orthogonal term graph rewriting. The latter is confluent, but lambda graph rewriting with cycles is in full generality inherently non-confluent. However, suitable restrictions can be formulated that ensure confluence.

The paper is organized as follows: we start (in Section 2) with a discussion of rewriting with μ-recursion, a related but less expressive framework; μ-expressions describe cyclic term graphs, but are not able to express sharing of common subterms. Our discussion of μ-recursion will provide an intuition why introducing the 'cycle-breaking' constant $\bullet$ is necessary. The next section (Section 3) presents our notational framework for systems of recursion equations. We establish that the bisimilarity class of a term graph (possibly with cycles, as all graphs in this paper) is a complete lattice, partially ordered by functional bisimulation. In the following section (Section 4), using Equational Logic reasoning, we characterize the fundamental notions of copying, substitution, flattening, and hiding. In Section 5, a translation between μ-expressions and recursion systems is discussed. Using this translation in Section 6 a sound and complete proof system with respect to the semantics given by infinite tree unwinding is given. In Section 7, the notions of redex, reduction and a characterization of rules are introduced; the concept of orthogonality for term graph rewriting is presented. A system without ambiguous rules is shown to be confluent. Moreover, if non-left-linear rules are also discarded (*i.e.,* for orthogonal term graph rewriting) it is shown that confluence holds even if copying is admitted. The observation that copying does not destroy confluence was already shown in [Sme93] for acyclic terms. In Section 8, a translation of a term rewriting system into its corresponding term graph rewriting system is presented. We end the paper with directions for future work.

2. Term rewriting with μ-recursion

Although we will deal mostly with rewriting of systems of recursion equations, we start with the related format of μ-expressions and the μ-rule. This is done for two reasons: first, to appreciate the extra expressive power that recursion equations have above μ-expressions, and second, because the μ-formalism will provide us with a good intuition on how to solve the problem of *cyclic collapsing contexts* that constitute a problem for confluence.

2.1 Orthogonal TRSs with the μ-rule

Definition 2.1 Let R be a (first-order) TRS. Then R_μ results from R by adding the μ-rule:

$$\mu x.Z(x) \longrightarrow Z(\mu x.Z(x)).$$

Usually this rewriting rule is rendered as $\mu x.Z \longrightarrow Z[x \setminus \mu x.Z]$, where $[\ \setminus\ ]$ is the substitution operator. Here we use the notation as used for 'higher-order term rewriting' by means of Combinatory Reduction Systems (CRSs), as in [Klo, KvOvR93, vR93].

Proposition 2.2 *Let R be an orthogonal TRS. Then R_μ is an orthogonal CRS, and hence confluent.*

Proof: It is simple to check that R_μ is an orthogonal CRS, hence the general confluence theorem for orthogonal CRSs applies. See *e.g.,* [vR93]. □

Remark 2.3

(i) Orthogonality is not necessary to guarantee confluence. In fact:
Let R be a left-linear, confluent TRS. Then R_μ is a confluent CRS.

(ii) In (*i*) left-linearity is necessary. Otherwise there is the following counterexample:

$$\begin{array}{lcl} \mathsf{S}(x){-}\mathsf{S}(y) & \longrightarrow & x{-}y \\ \mathsf{0}{-}x & \longrightarrow & \mathsf{0} \\ x{-}\mathsf{0} & \longrightarrow & x \\ x{-}x & \longrightarrow & \mathsf{0} \\ \mathsf{S}(x){-}x & \longrightarrow & \mathsf{S}(\mathsf{0}) \end{array}$$

This a confluent TRS defining cut-off subtraction on natural numbers. However, R_μ is not confluent: let ω be $\mu x.\mathsf{S}(x)$, then $\omega \longrightarrow \mathsf{S}(\omega)$, and

$$\begin{tikzcd} \omega{-}\omega \arrow[r] \arrow[d] & \mathsf{0} \\ \mathsf{S}(\omega){-}\omega \arrow[r] & \mathsf{S}(\mathsf{0}) \end{tikzcd}$$

Example 2.4 Let R be the orthogonal TRS with the two collapsing rules: $\mathsf{A}(Z) \longrightarrow Z, \mathsf{B}(Z) \longrightarrow Z$, then R_μ is the orthogonal CRS with rules:

$$\begin{array}{lcl} \mathsf{A}(Z) & \longrightarrow & Z \\ \mathsf{B}(Z) & \longrightarrow & Z \\ \mu x.Z(x) & \longrightarrow & Z(\mu x.Z(x)) \end{array}$$

Now we have the reductions:

$$\begin{array}{c} \mu x.\mathsf{A}(\mathsf{B}(x)) \longrightarrow \mu x.\mathsf{A}(x) \longrightarrow \mu x.x \\ \mu x.\mathsf{A}(\mathsf{B}(x)) \longrightarrow \mu x.\mathsf{B}(x) \longrightarrow \mu x.x \end{array}$$

The expression $\mu x.x$ plays an important role, and we will abbreviate it by $\bullet$. Note that the term $\mu x.\mathsf{A}(\mathsf{B}(x))$ corresponds to the infinite term $(\mathsf{AB})^\omega$, and the reduction $\mu x.\mathsf{A}(\mathsf{B}(x)) \longrightarrow \mu x.\mathsf{A}(x)$ corresponds to the infinite reduction $(\mathsf{AB})^\omega \longrightarrow_\omega \mathsf{A}^\omega$. Analogously, we have $(\mathsf{AB})^\omega \longrightarrow_\omega \mathsf{B}^\omega$. However, note that the μ-calculus is able to perform the reduction $\mu x.\mathsf{A}(x) \longrightarrow \bullet$, while the infinitary calculus can only reduce A^ω to itself, and likewise B^ω, thus violating confluence.

Example 2.5 Let CL be "Combinatory Logic", with the rules:

$$\begin{aligned}&\mathsf{S}Z_1Z_2Z_3 \longrightarrow Z_1Z_3(Z_2Z_3)\\&\mathsf{K}Z_1Z_2 \longrightarrow Z_1\\&\mathsf{I}Z \longrightarrow Z\end{aligned}$$

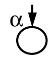
Then CL_μ ("Combinatory Logic with μ-recursion") is the orthogonal CRS obtained by adding the μ-rule. CL_μ is confluent.

2.2 Connection with term graphs

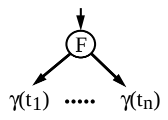
Anticipating a more precise treatment of term graphs in the next section, and a precise translation of μ-terms into recursion systems (*i.e.,* term graph) in Section 5, we now describe the assignment of a term graph to a μ-term. To that end, let us review the formation of μ-terms over a signature Σ.

Definition 2.6 Let Σ be a first-order signature. Then $\mathrm{Term}(\Sigma_\mu)$, the set of μ-terms over Σ, is given by:
(i) $\alpha, \beta, \gamma, \cdots \in \mathrm{Term}(\Sigma_\mu)$ (variables);
(ii) $t_1, \cdots, t_n \in \mathrm{Term}(\Sigma_\mu) \Longrightarrow \mathsf{F}(t_1, \cdots, t_n) \in \mathrm{Term}(\Sigma_\mu)$, for an n-ary function symbol in Σ;
(iii) $t \in \mathrm{Term}(\Sigma_\mu) \Longrightarrow \mu\alpha.t \in \mathrm{Term}(\Sigma_\mu)$.

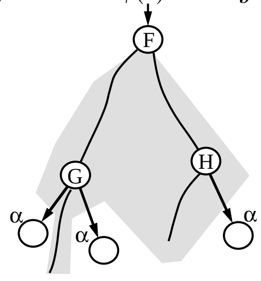
Definition 2.7 Let $t \in \mathrm{Term}(\Sigma_\mu)$. Then the term graph $\gamma(t)$ is defined as follows:
(i) $\gamma(\alpha)$ is:



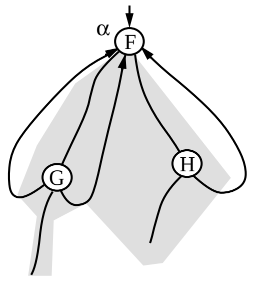
(ii) $\gamma(\mathsf{F}(t_1, \cdots, t_n))$ is :



(iii) $\gamma(\mu\alpha.t)$: if α does not occur free in t, then $\gamma(\mu\alpha.t) = \gamma(t)$.
Suppose t contains a free α and $t \not\equiv \alpha$. Let $\gamma(t)$ be *e.g.,*

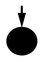


(where all the free α's are displayed) then $\gamma(\mu\alpha.t)$ is:



In case the root node has already a name, say β, we will replace it by α.

(iv) $\gamma(\mu\alpha.\alpha)$ is :



Working in R_μ is already a form of term graph rewriting. The reductions in Example 2.4 and Example 2.5 are shown pictorially in Figure 3 and Figure 4, respectively.

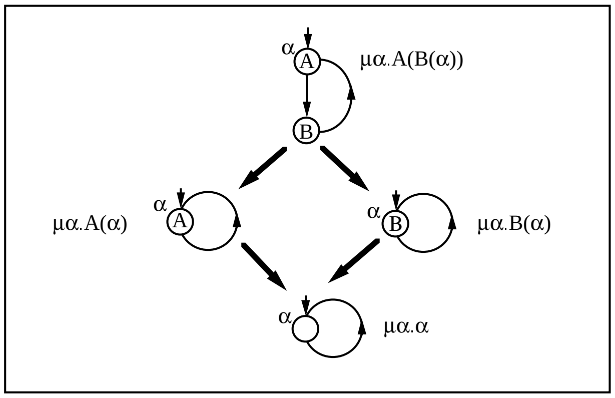


Figure 3.

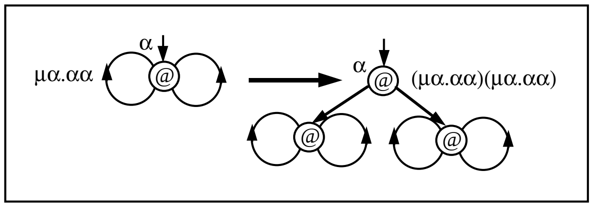


Figure 4.

Example 2.8 Let $SKIM$ be as in Table 1. Let $SKIM^*$ be $SKIM$ minus the rule $\mathsf{Y}x \longrightarrow x(\mathsf{Y}x)$ and extended with the μ-rule and the following the new version of the Y rule:

$$\mathsf{Y}Z \longrightarrow \mu x.Zx$$

$\mathsf{S}xyz$	$\longrightarrow$	$xz(yz)$	
$\mathsf{K}xy$	$\longrightarrow$	x	
$\mathsf{I}x$	$\longrightarrow$	x	
$\mathsf{C}xyz$	$\longrightarrow$	xzy	
$\mathsf{B}xyz$	$\longrightarrow$	$x(yz)$	
$\mathsf{Y}x$	$\longrightarrow$	$x(\mathsf{Y}x)$	
$\mathsf{U}z\mathsf{P}(xy)$	$\longrightarrow$	zxy	
$\mathsf{P}_0(\mathsf{P}xy)$	$\longrightarrow$	x	
$\mathsf{P}_1(\mathsf{P}xy)$	$\longrightarrow$	y	
$\mathsf{cond\ true}\ x\ y$	$\longrightarrow$	x	
$\mathsf{cond\ false}\ x\ y$	$\longrightarrow$	y	
$\mathsf{plus}\ \underline{n}\ \underline{m}$	$\longrightarrow$	$\underline{n+m}$	
$\mathsf{times}\ \underline{n}\ \underline{m}$	$\longrightarrow$	$\underline{n*m}$	
$\mathsf{eq}\ \underline{n}\ \underline{n}$	$\longrightarrow$	$\underline{true}$	
$\mathsf{eq}\ \underline{n}\ \underline{m}$	$\longrightarrow$	$\underline{false}$	if $n \neq m$

Table 1. SKIM

which pictorially looks as in Figure 1. $SKIM^*$ is an orthogonal CRS, hence confluent. Actually, this is the way that $SKIM$ was implemented.

The shortcoming of μ-expressions as in R_μ is that while some cyclic graphs can be represented as such, many cannot, *e.g.,* the graphs in Figure 5 cannot be represented by a μ-expression. Roughly said, μ-expressions only describe "vertical sharing" (see Figure 6) and not "horizontal sharing". We say that a graph *has only vertical sharing* if the graph can be partitioned into a tree and a set of edges with the property that either begin and end nodes are identical, or the end node is an ancestor (in the tree) of the begin node. (We will come back to this distinction in Section 3.4.) Equivalently, a graph has only vertical sharing if there are no two different acyclic paths starting from the root to the same node.

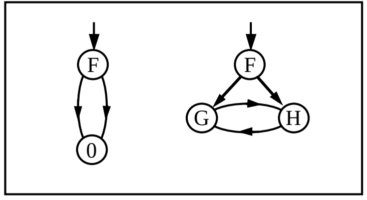


Figure 5. Horizontal sharing

Example 2.9 The μ-term corresponding to the graph in Figure 6 is

$$\mu\alpha.\mathsf{F}(\mu\beta.\mathsf{G}(\mathsf{G}(\alpha,\beta),\mathsf{H}(0)),\mathsf{H}(\mathsf{H}(\mathsf{H}(\alpha))),\mu\gamma.\mathsf{G}(0,\mathsf{G}(\alpha,\mathsf{H}(\gamma)))).$$

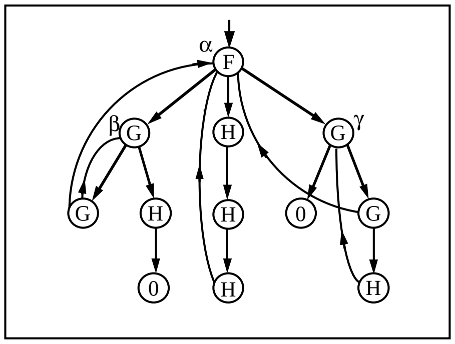


Figure 6. Vertical sharing

3. Systems of recursion equations

In this section we will consider systems of recursion equations, establish a notational framework for them, and study some of their properties. Specifically we introduce the notion of bisimilarity of systems of recursion equations. This notion is well-known from the theory of processes (or 'concurrency' or 'communicating processes' or 'process algebra') - see Milner [Mil89], Baeten & Weijland [BW90]. We establish lattice properties of the bisimilarity equivalence class, and show that checking whether two graphs are bisimilar can be done in an equational way.

3.1 Syntax of systems of recursion equations

Our notation for graphs comes from the intuition that a natural way of linearly representing a graph is by associating a unique name to each node and by writing down the interconnections through a set of recursion equations. For example, we will represent the graph depicted in Figure 7 as follows:

$$\{\alpha = \mathsf{F}(\beta,\gamma),\ \beta = \mathsf{G}(\alpha),\ \gamma = \mathsf{H}(\alpha,\alpha)\}$$

where we assume that the first recursion variable represents the root of the graph. Alternatively, we sometimes write the above as:

$$\{\alpha \mid \alpha = \mathsf{F}(\beta,\gamma),\ \beta = \mathsf{G}(\alpha),\ \gamma = \mathsf{H}(\alpha,\alpha)\}.$$

Systems of recursion equations are considered modulo renaming of the (bound) recursion variables. *E.g.,* $\{\delta = \mathsf{F}(\epsilon,\eta),\ \epsilon = \mathsf{G}(\delta),\ \eta = \mathsf{H}(\delta,\delta)\}$ is equivalent to the above system. Similar notations appear in the literature [GKS90, Far90]. *E.g.,* $\{u : \mathsf{F}(u,v),\ v : \mathsf{G}(u)\}$ in the language DACTL [GKS90]. However, *we insist on an equational notation*, not just for the sake of style, but because Equational Logic reasoning can fruitfully support our thinking about common term graph operations, as will become clear shortly.

Definition 3.1 Let Σ be a first-order signature.
(i) The set of TRS terms over Σ ($Term(\Sigma)$) is given by:
  (i.1) $\alpha,\beta,\gamma,\cdots \in Term(\Sigma)$ (variables);
  (i.2) if $t_1,\cdots,t_n \in Term(\Sigma)$ then $\mathsf{F}(t_1,\cdots,t_n) \in Term(\Sigma)$.
(ii) A *system of recursion equations* over Σ is of the form

$$\{\alpha_1 = t_1,\cdots,\alpha_n = t_n\}$$

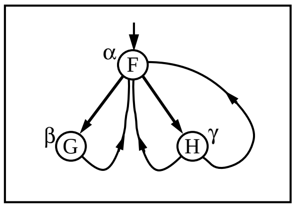


Figure 7.

with $t_1, \cdots, t_n \in \text{Term}(\Sigma)$, $\alpha_1, \cdots, \alpha_n$ recursion variables such that $\forall i, j, 1 \leq i < j \leq n$, $\alpha_i \not\equiv \alpha_j$. The variables α_i are bound; other variables occurring in the system are free. A system without free variables is closed.

We do not have nesting of recursion equations as in [Ari92], see however Section 9. Moreover, multiple definitions of a variable are not allowed. *E.g.,* $\{\alpha = \mathsf{3},\ \alpha = \mathsf{4}, \cdots\}$ is not a well-formed system. Unless otherwise stated we only consider systems without useless equations; an equation is useless if its leading recursion variable is not reachable from the root of the system, in the obvious sense. We automatically perform this removal (garbage collection).

We call a system of recursion equations in *flattened form* if the right-hand side of each equation is of the form $\mathsf{F}(\alpha_1, \cdots, \alpha_n)$, where the α_i are recursion variables, not necessarily distinct from each other. *E.g.,* $\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\alpha)\}$ is in flattened form, while $\{\alpha = \mathsf{F}(\mathsf{G}(\alpha))\}$ is not. The distinction between the two terms above will become clear after having introduced (in Section 4) the substitution operation. A flattened system is in *canonical* form if it does not contain trivial equations of the form $\alpha = \beta$; such equations also if not present in the original term can arise as a result of a reduction step. We perform the substitution of each occurrence of α by β and the removal of the corresponding equation as part of a canonicalization step. An equation of the form $\alpha = \alpha$ will be replaced by $\alpha = \bullet$.

Notation: we denote by $M \mid \beta$ the subsystem rooted at β. *E.g.,* let M be $\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\gamma)\}$, then $M \mid \beta$ is $\{\beta = \mathsf{G}(\gamma)\}$; the equation $\alpha = \mathsf{F}(\beta)$ gets removed.

3.2 Correspondence with terms graphs

Term graphs can be described in many ways. The usual way is to equip a set of nodes and a set of edges with several functions. *E.g.,*

Definition 3.2 Let Σ be a first-order signature, with $Fun(\Sigma)$ the set of functions and constant symbols. Then *a term graph* over Σ, is a structure

$$\langle N, Lab, Succ, root\rangle$$

with:
(i) N a set of nodes;
(ii) $Lab: N \rightarrow Fun(\Sigma) \cup \perp$;
(iii) $Succ: N \rightarrow N^*$, such that for all $s \in N$, if n is the arity of $Lab(s)$, then $Succ(s)$ must be a n-tuple of nodes;
(iv) $root \in N$.

An important point of this paper is that such definitions can be avoided and that we can do everything with systems of recursion equations. However, for an intuitive and quick grasp of such a system, it is for human consumption often convenient to draw the corresponding graph, as we will do many times in this paper.

We will now give a definition of term graph independent of a system of recursion equations, slightly different from the one given above, using the concept of a structure in model theory. Our notion of term graph is more complicated than the usual one, since we employ names of nodes (to be distinguished from the label of a node), and also admit a partial naming (some nodes named, others nameless). Actually, we employ equivalence classes of such objects, identifying objects that can be obtained from each other by 1-1 renaming (cf. α-conversion in lambda calculus).

First we define a pseudo-term graph. This is a first-order structure (in the sense of model theory) of the form

$$G = \langle \mathtt{NODES}, \mathtt{ROOT}, \alpha_1, ..., \alpha_k, a_1, ..., a_m \rangle$$

Here NODES is some set of elements $s, t, \cdots$ called nodes, ROOT is a unary predicate, α_i ($i = 1, \cdots, k$) are binary predicates, and $a_1, \cdots, a_m$ are constants. Elements satisfying the ROOT predicate are called roots, and constants are also called node names. Now let Σ be a first-order signature, with set of function symbols $Fun(\Sigma)$. We define: a Σ-term graph is a pseudo-term graph as above together with a partial map

$$content : \mathtt{NODES} \to Fun(\Sigma)$$

such that if $content(s)$ is an n-ary function symbol, s has an i^{th}-successor for every $i = 1, \cdots n$ and no more, and if $content(s)$ is undefined, s has no i-successor. In the last case, s is called an empty node.

In fact, we are interested in Σ-term graphs modulo renaming. We define $(G, content)$ and $(G', content')$ to be α-equivalent, if G and G' are isomorphic in the model-theoretic sense, and the content mappings commute with this isomorphism. Now the objects of interest are the α-equivalence classes.

This definition is still too general for our purpose; it admits multiple roots, or none at all, it admits totally unnamed graphs, and empty nodes without a name; it does not require the graph to be connected; but it is easy to formulate some extra requirements so that we have an exact match with systems of recursion equations. We will not do so, as we will not need this definition of graph. However, it should be noted that this definition (or similar variants like the one above) is an unnecessary circumlocution; a system of recursion equations is much easier defined, leaving notions as i-successor implicit. Explicit definitions like the one above do not seem to help the intuition much.

There is another way of introducing term graphs that we find much more helpful for an intuitive grasp, even though it may seem a bit too fancy at first sight. This is by perceiving the 'underlying space' in which a term is written, as Baire space, and then considering the homomorphic images of this space obtained by identifying nodes. Terms written in such a homomorphic image are then just term graphs. We will give the precise definitions in Section 3.4.

Hereafter we allow ourselves to interchangeably use the words: system of recursion equations or (term) graph.

We now introduce some notation that will be needed in the next section: given a graph g we will denote by $\mathtt{ROOT}(g)$ and $\mathtt{NODES}(g)$ the root and the set of nodes in g; if s is a node in a term graph g and s' is its i^{th}-successor, then we will write $s \rightarrow_i s'$.

Definition 3.3 An *access path* of a node s of a term graph g is a possibly empty finite sequence of positive natural numbers $\langle i_1, i_2, \cdots, i_j \rangle$ such that there exist nodes $s_1, \cdots, s_{j-1}$ in g with $\mathtt{ROOT}(g) \rightarrow_{i_1} s_1 \rightarrow_{i_2} \cdots \rightarrow_{i_{j-1}} s_{j-1} \rightarrow_{i_j} s$.

In general a node s may have several access paths; we will denode by $\mathtt{Acc}(s)$ the set of the access paths of s (the empty access path $\langle\ \rangle$ denotes one of the access paths to the root). We will call a term graph g *connected* if the access set of each node in g is not empty. Notation: given access paths π_1 and π_2, $\pi_1 * \pi_2$ denotes the concatenation of π_1 and π_2.

Remark 3.4 The access sets express various properties of graphs:
(i) if every node s in g has a singleton as $\mathtt{Acc}(s)$ then g is a tree;
(ii) if for all nodes s in g, $\mathtt{Acc}(s)$ is finite then g is a dag;
(iii) if there exists a node s in g, with $\mathtt{Acc}(s)$ infinite then g is a cyclic graph, provided g is a finite graph.

3.3 Bisimulations

In this section, we will restrict our attention to closed systems of recursion equations in flattened form only. At the end of this section we will discuss how to cover the non-flat case.

Definition 3.5 Let $g = \{\alpha_0 = t_0, \cdots, \alpha_n = t_n\}$, and let $h = \{\alpha'_0 = t'_0, \cdots, \alpha'_m = t'_m\}$. Then R is a *bisimulation* from g to h if
(i) R is a binary relation with domain $\{\alpha_0, \cdots, \alpha_n\}$ and codomain $\{\alpha'_0, \cdots, \alpha'_m\}$;
(ii) $\alpha_0 \ R \ \alpha'_0$ (the roots are related);
(iii) if $\alpha_i \ R \ \alpha'_j$,
$\alpha_i = \mathsf{F}_i(\alpha_{i1}, \cdots, \alpha_{ik})$ $(k \geq 0)$
$\alpha'_j = \mathsf{F}_j(\alpha'_{j1}, \cdots, \alpha'_{jk'})$ $(k' \geq 0)$ then $\mathsf{F}_i \equiv \mathsf{F}_j$, $k = k'$, and $\alpha_{i1} \ R \ \alpha'_{j1}, \cdots, \alpha_{ik} \ R \ \alpha'_{jk'}$
So, related nodes have the same label, and their successor nodes are again related. (Notation: we will interchangeably use the notation $\alpha \ R \ \alpha'$ or $(\alpha, \alpha') \in R$.)

Definition 3.6
(i) Graphs g,h are *bisimilar* if there is a bisimulation from g to h. We will write: $g \underline{\leftrightarrow} h$.
(ii) $g \underline{\rightarrow} h$ if there is a *functional bisimulation* from g to h, *i.e.,* a bisimulation that is a function.

Remark 3.7 Bisimilarity is an equivalence relation.

A functional bisimulation is what in the literature [BvEG$^+$87, Sme93] is called a rooted homomorphism; it collapses (compresses) the graph to a smaller one. Vice versa we say that the graph is 'expanded', and this is the 'copying' or 'unsharing' or 'unwinding' partial order appearing in [Ari92, Ari93, AA93]. See Figure 8, where some unwindings of the graph $\{\alpha = \mathsf{F}(\alpha)\}$ are considered. We use the notation (n, m) to indicate the unwinding of this graph starting with n 'acyclic steps' followed by a cycle of m steps $(n \geq 0, m \geq 1)$.

The importance of the notion of bisimilarity stems from the fact that bisimilarity corresponds to having the same tree unwinding (*i.e.,* same semantics). We need a proposition first:

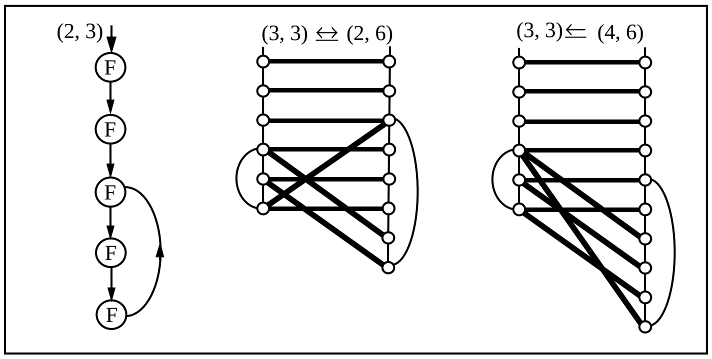


Figure 8.

Proposition 3.8
(i) *Let g be a graph, $[\![g]\!]$ its tree unwinding. Then: $g \underline{\leftarrow} [\![g]\!]$.*
(ii) *Let g, h be trees (finite or infinite). Then: $g \underline{\leftrightarrow} h \Longleftrightarrow g = h$.*

Proof:
(i) It is easy to see that $[\![g]\!]$ can be obtained as follows:

$$\begin{array}{lcl} \texttt{ROOT}([\![g]\!]) & = & \langle\,\rangle \\ \texttt{NODES}([\![g]\!]) & = & \bigcup\{\texttt{Acc}(s) \mid s \in \texttt{NODES}(g)\} \\ i\text{-successor} & : & \text{if } s \rightarrow_i s' \text{ in } g, \text{ and } \pi \in \texttt{Acc}(s), \text{ then } \pi \rightarrow_i \pi * i. \end{array}$$

(Note that $\pi * i \in \texttt{Acc}(s')$.)
The content of each node in $[\![g]\!]$ contributed by $\texttt{Acc}(s)$ is the content of s.

Now define $\phi : \texttt{NODES}([\![g]\!]) \rightarrow \texttt{NODES}(g)$ by:

$$\text{if } \pi \in \texttt{Acc}(s), \text{ then } \phi(\pi) = s.$$

Checking that this yields $\phi : [\![g]\!] \underline{\rightarrow} g$, *i.e.,* ϕ is a functional bisimulation from $[\![g]\!]$ to g, is routine.

(ii) If g is a tree, let $(g)_n$ be the finite tree truncated at depth n (appending some constant, *e.g.,* Ω, at the cut-points). Now it is easy to prove from $g \underline{\leftrightarrow} h$ that $(g)_n = (h)_n$ for all n. Hence $g = h$ (More precisely, g and h are isomorphic.).

□

Corollary 3.9 *Let g, h be graphs. Then : $g \underline{\leftrightarrow} h \iff [\![g]\!] = [\![h]\!]$.*

Proof:
($\Longrightarrow$) Suppose $g \underline{\leftrightarrow} h$. By Proposition 3.8(i):

$$[\![g]\!] \underline{\leftrightarrow} g \underline{\leftrightarrow} h \underline{\leftrightarrow} [\![h]\!] .$$

By transitivity of $\underline{\leftrightarrow}$, $[\![g]\!] \underline{\leftrightarrow} [\![h]\!]$. By Proposition 3.8(ii): $[\![g]\!] = [\![h]\!]$.
($\Longleftarrow$) Suppose $[\![g]\!] = [\![h]\!]$. By Proposition 3.8:

$$g \underline{\leftrightarrow} [\![g]\!] \underline{\leftrightarrow} [\![h]\!] \underline{\leftrightarrow} h.$$

By transitivity of $\underline{\leftrightarrow}$, $g \underline{\leftrightarrow} h$.

□

We will show next that the equivalence class of a graph g with respect to the equivalence relation of bisimilarity, partially ordered by functional bisimilarity, is a complete lattice (*i.e.,* a partial order where every subset has a least upper bound (*lub*) and a greatest lower bound (*glb*)). For the acyclic case, the lattice property is proved in Smetsers [Sme93]. We expect that in maybe slightly different but related settings this is a well-known fact, but we include the following proof for completeness sake and also to demonstrate the use of the notion of bisimilarity.

Remark 3.10
(i) Let R_1, R_2 be two bisimulations from g to h. Then $R_1 \cap R_2$ is again a bisimulation from g to h.
(ii) Let $g \underline{\leftrightarrow} h$. Then there exists a unique minimal bisimulation from g to h. Notation: $R_{g,h}$. Clearly, the inverse relation $(R_{g,h})^{-1}$ is $R_{h,g}$.
(iii) Let $g \underline{\rightarrow} h$. Then $R_{g,h}$ is functional.

Proof: Directly from the definition of bisimulation. □

Notation: instead of R, we will denote a functional bisimulation also by ϕ.

Proposition 3.11 *Functional bisimilarity is a partial order.*

Proof: $\underline{\rightarrow}$ is clearly reflexive and transitive (because functional bisimulations are closed under composition). Now suppose $g \underline{\rightarrow} h \underline{\rightarrow} g$. Then, by Remark 3.10, the minimal bisimulations from g to h, and from h to g, respectively, are each other's inverse and moreover functional. It follows that they are bijections. In other words, g and h are rooted isomorphic. □

Definition 3.12 Let $R : g \underline{\leftrightarrow} h$ be a bisimulation. Then the *associated graph* γ_R is defined as follows:
(i) $\texttt{NODES}(\gamma_R) \;=\; R$
$\texttt{ROOT}(\gamma_R) \;=\; (\texttt{ROOT}(g), \texttt{ROOT}(h))$
the label of each node (s,t) is that of s (or, what is the same, of t) .
(ii) Let $s \in \texttt{NODES}(g)$, $t \in \texttt{NODES}(h)$, $(s,t) \in R$, $s \rightarrow_i s', t \rightarrow_i t'$.
Then in $\gamma_R : (s,t) \rightarrow_i (s',t')$.

Proposition 3.13
(i) *Let $R : g \underline{\leftrightarrow} h$. Then R is minimal $\Longleftrightarrow$ $(s,t) \in R$ implies* $\texttt{Acc}(s) \cap \texttt{Acc}(t) \neq \emptyset$.
(ii) *Let $R_{g,h} : g \underline{\leftrightarrow} h$, and $(s,t) \in \texttt{NODES}(\gamma_{R_{g,h}})$. Then* $\texttt{Acc}((s,t)) = \texttt{Acc}(s) \cap \texttt{Acc}(t)$.

Proof:

(i) Let $R : g \underline{\leftrightarrow} h$. By the definition of minimal bisimulation, it follows that whenever $s \in \texttt{NODES}(g)$, $t \in \texttt{NODES}(h)$ have a common access path, we have $(s,t) \in R$. Reversely, a pair (s,t) with common access path must be in every bisimulation from g to h. Hence it is in the minimal bisimulation, being the intersection of all bisimulations.

(ii) Let $(s,t) \in R_{g,h}$. A common access path of s,t obviously is an access path of (s,t) in $\gamma_{R_{g,h}}$, and vice versa.

□

Corollary 3.14 *Let $R : g \underline{\leftrightarrow} h$ be a bisimulation. Then: R is minimal $\iff \gamma_R$ is connected.*

Proof: R is minimal iff whenever $(s,t) \in R$ we have $\mathtt{Acc}(s) \cap \mathtt{Acc}(t) \neq \emptyset$ (Proposition 3.13(i)) iff whenever $(s,t) \in R$ we have $\mathtt{Acc}((s,t)) \neq \emptyset$ (Proposition 3.13(ii)) iff γ_R is connected. □

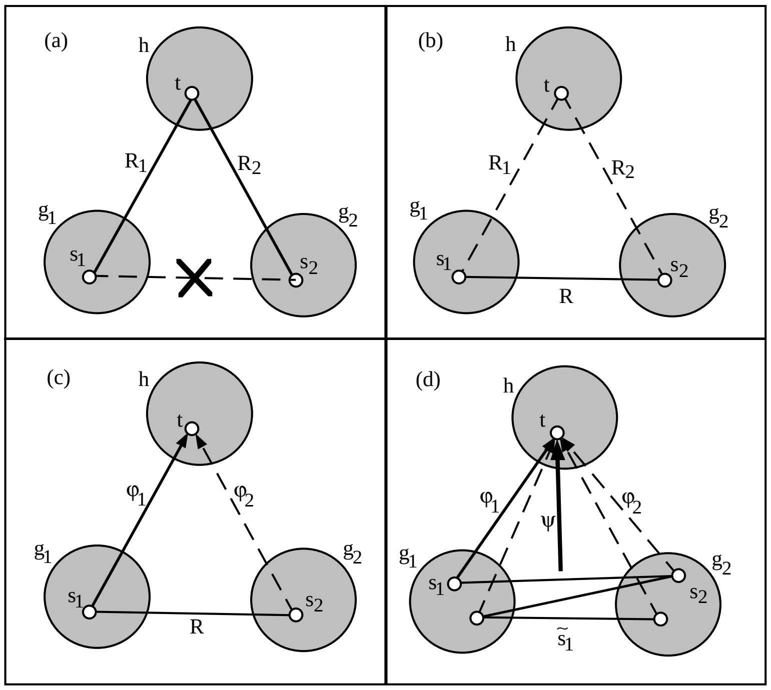


Figure 9.

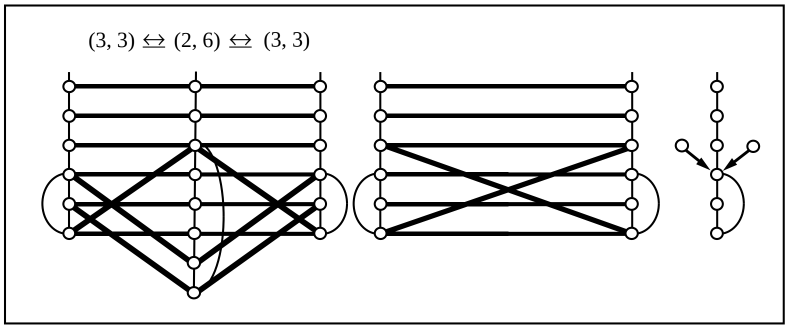


Figure 10.

Remark 3.15 Let $R_1 : g \underline{\leftrightarrow} h$ and $R_2 : h \underline{\leftrightarrow} r$ be two minimal bisimulations. Then the composition $R_1 \circ R_2 : g \underline{\leftrightarrow} r$ is again a bisimulation, but it needs not be minimal (see Figure 9(a)). As an example, let g be $(3,3)$, h be $(2,6)$ and $r = g$. See Figure 10; here the middle figure is the composition of the two bisimulations in the left figure. The right figure is the associated graph, which is not connected. (An example with r different from g, h is also easy to give.)

Proposition 3.16 *Let $R : g_1 \underline{\leftrightarrow} g_2$ be a minimal bisimulation, and likewise*

$$R_1 : g_1 \underline{\leftrightarrow} h, \quad R_2 : g_2 \underline{\leftrightarrow} h.$$

Let $(s_1, s_2) \in R$. Then there exists a $t \in h$ such that $(s_1, t) \in R_1$, $(s_2, t) \in R_2$.

Proof: By Proposition 3.13(i) there exists a common access path π to s_1 and to s_2. Let t be the node in h reached after the same access path π, then by applying again Proposition 3.13(i) we have that t must be related to s_1 via R_1 and to s_2 via R_2. (See Figure 9(b).) □

Proposition 3.17 *Let $R : g_1 \underline{\leftrightarrow} g_2$ be a minimal bisimulation and let $\phi_1 : g_1 \underline{\rightarrow} h$ and $\phi_2 : g_2 \underline{\rightarrow} h$ be minimal functional bisimulations. Suppose $(s_1, s_2) \in R$ and $\phi_1(s_1) = t$. Then, also $\phi_2(s_2) = t$. (See Figure 9(c).)*

Proof: Suppose, in addition to the assumptions of Proposition 3.16, that R_1 and R_2 are functional. Then t is unique. Hence the proposition follows. □

Before stating the main theorem of this section we introduce the following definition.

Definition 3.18 Let G be a set of bisimilar graphs. An *accessible fibre through G* is a choice function Ψ on G (*i.e.,* a function selecting an element of each $g \in G$)

$$\Psi : G \to \bigcup_{g \in G} \texttt{NODES}(g)$$

with $\Psi(g) \in \texttt{NODES}(g)$, such that:

$$\bigcap_{g \in G} \texttt{Acc}(\Psi(g)) \neq \emptyset.$$

In other words, Ψ can be obtained as end stage of a march in lock-step, simultaneously in all graphs $g \in G$.

Theorem 3.19 *The bisimilarity class of graph g, partially ordered by functional bisimulation, is a complete lattice.*

Proof: The proof will be given in four parts:
(i) Given $g_1 \underline{\leftrightarrow} g_2$, we show the existence of $g_1 \vee g_2$ (*i.e.,* the join).
(ii) Given $g_1 \underline{\leftrightarrow} g_2$, we show the existence of $g_1 \wedge g_2$ (*i.e.,* the meet).
(iii) Given a set G of bisimilar graphs, we show the existence of $\bigvee G$.
(iv) Likewise for $\bigwedge G$.

(i) Let $R : g_1 \underline{\leftrightarrow} g_2$ be a minimal bisimulation. Then the associated graph γ_R is in fact $g_1 \vee g_2$. To see that $\gamma_R \underline{\rightarrow} g_1$ and $\gamma_R \underline{\rightarrow} g_2$, take the projection maps:

$$\begin{aligned} p_1 :& \quad (s_1, s_2) \mapsto s_1 \\ p_2 :& \quad (s_1, s_2) \mapsto s_2 \end{aligned}$$

It is easy to see that these are functional bisimulations as required. Moreover, if $h \underline{\rightarrow} g_1$ and $h \underline{\rightarrow} g_2$, then $h \underline{\rightarrow} \gamma_R$. Namely, if $t \in h$, $t \mapsto s_1 \in g_1$ and $t \mapsto s_2 \in g_2$, we define: $t \mapsto (s_1, s_2)$ and this is the required functional bisimulation from h to γ_R. (See Figure 11.)

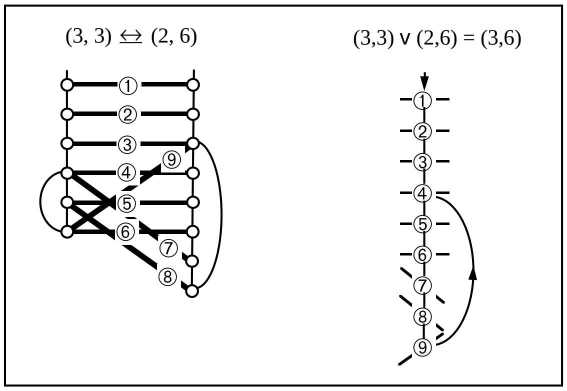


Figure 11.

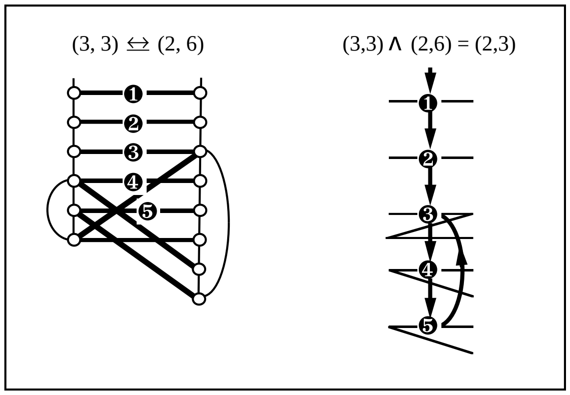


Figure 12.

(ii) Let $g_1 \underline{\leftrightarrow} g_2$. Again the minimal bisimulation R from g_1 to g_2 enables a simple construction of $g_1 \wedge g_2$. Let $\sim$ be the equivalence relation induced by R on $\texttt{NODES}(g_1) \cup \texttt{NODES}(g_2)$. The equivalence classes $\tilde{s_1}$ ($s_1 \in \texttt{NODES}(g_1)$) will be the nodes of a graph called δ_R. If s_0 is $\texttt{ROOT}(g_1)$, then $\tilde{s_0}$ is $\texttt{ROOT}(\delta_R)$. For i-successor we define: if $s \rightarrow_i s'$, then $\tilde{s} \rightarrow_i \tilde{s'}$. We claim that δ_R is $g_1 \wedge g_2$. To show $g_1 \underline{\rightarrow} \delta_R$ and $g_2 \underline{\rightarrow} \delta_R$, take $s_1 \mapsto \tilde{s_1}$ and $s_2 \mapsto \tilde{s_2}$. It is easy to show that these are functional bisimulations ϕ_1, ϕ_2, as required.
Moreover to show : if $g_1 \underline{\rightarrow} h$ and $g_2 \underline{\rightarrow} h$, then $\delta_R \underline{\rightarrow} h$. So take $\tilde{s_1} \in \delta_R$. Let $\phi_1 : g_1 \underline{\rightarrow} h$ be the minimal bisimulation. Let $\phi_1(s_1) = t$. Then define $\psi(\tilde{s_1}) = t$. Well-definedness follows from Proposition 3.17, which entails that all elements in $\tilde{s_1}$ are sent to t. (See Figure 12 and 9(d).)

(iii) $\bigvee G$ is the graph with as nodes the accessible fibres through G. It is clear how to define the root of $\bigvee G$, and how to define the successor relations.

(iv) On $V = \bigcup_{g \in G} \texttt{NODES}(g)$ we define: for $s_1 \in \texttt{NODES}(g_1)$, $s_2 \in \texttt{NODES}(g_2)$: $s_1 \sim_m s_2$ if s_1, s_2 are related by the minimal bisimulation between g_1, g_2. As noted before (Remark 3.15), $\sim_m$ is not yet an equivalence relation on V, by the failure of transitivity. Let $\sim$ be the equivalence relation on V generated by $\sim_m$. Then the graph $\bigwedge G$ will have as nodes: the $\sim$-equivalence classes. Root and successor relations are defined as before, and verifying that the graph is indeed $\bigwedge G$ is as in (ii).

□

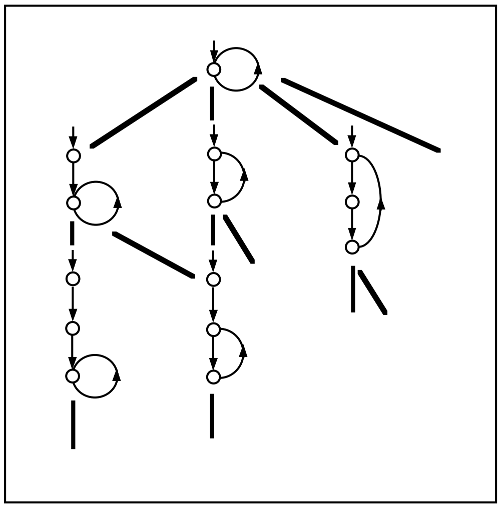


Figure 13.

Example 3.20 (i) Already the simplest cyclic graph, $\{\alpha = \mathsf{F}(\alpha)\}$, has a non-trivial complete lattice of expansions (see Figure 13). In fact, this lattice is isomorphic to the lattice:

$$(N^+ \cup \{\infty\}, |)$$

where N^+ is the set of positive natural numbers, and the partial order $|$ is defined by:

$$\begin{array}{lll} n & | & m \quad \text{if } n \text{ divides } m \\ n & | & \infty \\ \infty & | & \infty \end{array}$$

This can be seen as follows. As said before, let (n,m) , for $n \geq 0, m \geq 1$, be the graph starting with n 'acyclic steps' followed by a cycle of m steps. Let ∞ be the infinite unwinding of $\{\alpha = \mathsf{F}(\alpha)\}$. Now we have (see Figure 8):

$$\begin{array}{lll} (n,m) & \sqsupseteq & (n',m') \quad \text{iff } n' \leq n' \text{ and } m' \mid m \\ \infty & \sqsupseteq & \infty \\ \infty & \sqsupseteq & (n,m) \quad\;\; \text{for all } n \geq 0, m \geq 1 \end{array}$$

It is not hard to see that the first component n of (n,m) does not make the lattice more complicated than the one of positive natural numbers with divisor relation, as follows. Let $p_1, p_2, p_3, \cdots$ be the sequence of prime numbers $2, 3, 5, \cdots$. Let m have the prime decomposition:

$$m = p_1^{e_1}.p_2^{e_2}.p_3^{e_3}.\cdots$$

(An infinite product; all but finitely many e_i are 0.)
Define $\mathsf{shift}(m) = p_2^{e_1}.p_3^{e_2}.p_4^{e_3}.\cdots$. Now define :

$$\phi(n,m) = 2^n.\mathsf{shift}(m).$$

Then, $(n,m) \sqsupseteq (n',m')$ iff $\phi(n',m') \mid \phi(n,m)$. This proves the isomorphism with $(N^+ \cup \{\infty\}, |)$. Finally, note that $\vee$ and $\wedge$ are given by:

$$\begin{array}{lllll} (n,m) & \vee & (n',m') & = & (max(n,n'), lcm(m,m')) \\ (n,m) & \wedge & (n',m') & = & (min(n,n'), gcd(m,m')) \end{array}$$

(ii) The complete lattice of $\{\alpha = \mathsf{F}(\alpha,\beta), \beta = \mathsf{C}\}$ is even more complicated: it contains a sublattice of infinite elements of cardinality continuum. In fact, the sublattice of infinite elements is isomorphic to the lattice of partitions of N, the set of natural numbers. Figure 14, displaying from left to right respectively, the bottom, an intermediate element, the top of this lattice, suggests why this is so.

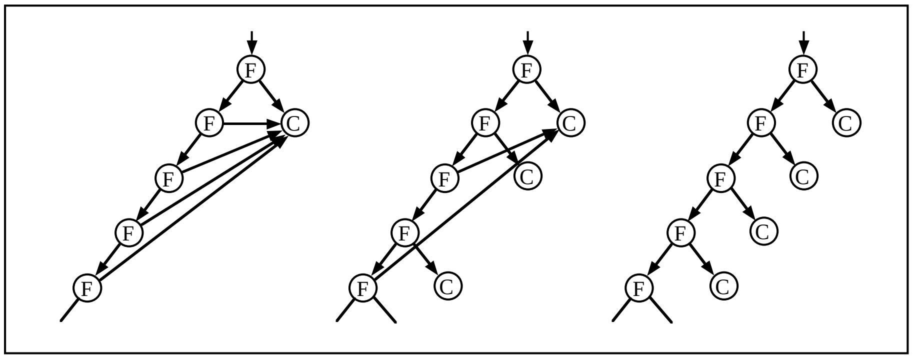


Figure 14.

Remark 3.21 Note the generality of the copying mechanism, in contrast with the restricted copying mechanism embodied in the μ-rule (the notion of copying will be defined precisely below). The example above with instead of $\{\alpha = \mathsf{F}(\alpha)\}$ the μ-term $\mu\alpha.\mathsf{F}(\alpha)$, would yield a sublattice isomorphic with $(N \cup \{\infty\}, <)$: $\mu\alpha.\mathsf{F}(\alpha) \longrightarrow \mathsf{F}(\mu\alpha.\mathsf{F}(\alpha)) \longrightarrow \cdots \longrightarrow \mathsf{F}^n(\mu\alpha.\mathsf{F}(\alpha)) \longrightarrow \cdots$

Remark 3.22 Our notation $g \underline{\leftarrow} h$ (g expands to h; g is a homomorphic image of h) intends to be reminiscent of the usual partial order symbol $\leq$, so that : information of $g \leq$ information of h. (It also suggests that h has more nodes than g.) The question is what 'information' is meant here. The answer is that h contains more *history information* than g. (Here a 'history' is the same as an access path.) Namely: suppose s, t are nodes in g, h respectively, such that s, t are related in a minimal bisimulation $R : g \underline{\leftarrow} h$. Then we have: $\mathtt{Acc}(t) \subseteq \mathtt{Acc}(s)$. (In fact, if $t_1, \cdots, t_n$ are all the nodes related to s, we have $\mathtt{Acc}(t_1) \cup \cdots \cup \mathtt{Acc}(t_n) = \mathtt{Acc}(s)$.) That is, we have sharper information about how we came to arrive, from the root, in t. Indeed, this is the reason why (in the setting of concurrency theory) a functional bisimulation is called a *history relation* in Lynch & Vaandrager [LV93]. The tree unwinding of a graph has maximum information; it is the top of the lattice. Each node in the tree has a singleton as history set.

Intermezzo 3.23 Our use of the notion of bisimulation in the present setting was suggested by process algebra (or 'concurrency'). Having established that functional bisimilarity is a partial order on term graphs, the question arises whether the same holds for process graphs. The situation there is more complicated, because of the laws $x + y = y + x$ and $x + x = x$ that process graphs modulo bisimilarity satisfy. That is, edges leaving a node are unordered, and bisimilar nodes need not have the same out-degree (*i.e.,* number of edges departing from them). Another difference with term graphs is that in process graphs the nodes are unlabeled, but the edges are. This difference is not essential for the present question, however. Figure 15

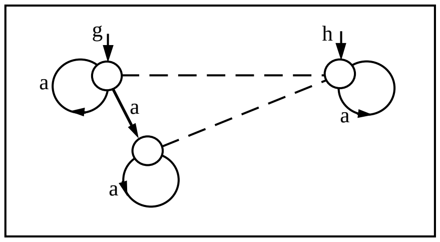


Figure 15.

displays a functional bisimulation between two process graphs g and h. Somewhat surprising, the situation for process graphs is now that for *finitely branching* process graphs, functional bisimilarity is also a partial order. The proof is more complicated than the easy one for term graphs. For infinitely branching process graphs, functional bisimilarity is only a pre-order. Figure 16 displays two infinitely branching process graphs, functionally bisimulating each

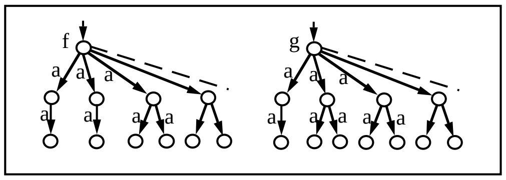


Figure 16.

other, yet different. The functional bisimulations are given by replacing the underlined parts as follows:

$$\begin{array}{lllllllll} f & = & \underline{aa+aa} & + & a(a+a) & + & a(a+a) & + & a(a+a) & + \cdots \\ g & = & aa & + & a(\underline{a+a}) & + & a(a+a) & + & a(a+a) & + \cdots \\ f & = & aa & + & aa & + & a(a+a) & + & a(a+a) & + \cdots \end{array}$$

Another important difference with process graphs is that there minimal bisimulations are in general not unique (*e.g.,* the term $a+a$ admits two minimal bisimulations with itself).

So far we have restricted our attention to flat systems only. We extend the notion of bisimulation to non-flat systems in two ways.

Weak bisimulation: Let us first introduce the notion of flattening, which is properly defined in Section 4. For now it suffices to say that `flat` is a function that rewrites an equation of the form $\alpha = \mathsf{F}^n(t_1, \cdots, t_n)$, $n \geq 0$, to $\alpha = \mathsf{F}(\alpha_1, \cdots, \alpha_n)$, and adds to the system the equations $\alpha_i = t_i$, $i = 1, \cdots, n$, where $\alpha_1, \cdots, \alpha_n$ are new names. *E.g.,* $\mathtt{flat}(\{\alpha = \mathsf{F}(\mathsf{G}(0), \mathsf{H}(1))\}) = \{\alpha = \mathsf{F}(\beta, \eta), \beta = \mathsf{G}(\gamma), \gamma = 0, \varepsilon = \mathsf{H}(\psi), \psi = 1\}$.

Definition 3.24 Graphs g and h are *weakly bisimilar* if $\mathtt{flat}(g) \underline{\leftrightarrow} \mathtt{flat}(h)$.

Strong bisimulation: We introduce a stronger notion of bisimulation, written as $\underline{\leftrightarrow}_s$, on non-flat systems by using a different mechanism to flatten a term, consisting in introducing new function symbols which memorize the structure of a term. Given for example the term $g \equiv \{\alpha = \mathsf{F}(\mathsf{H}(\mathsf{B}(\alpha)), \mathsf{G}(\alpha))\}$, instead of flattening g as $\{\alpha = \mathsf{F}(\alpha', \alpha''), \alpha' = \mathsf{H}(\alpha'''), \alpha'' = \mathsf{G}(\alpha), \alpha''' = \mathsf{B}(\alpha)\}$, we introduce a new function symbol $\mathsf{H}_{1\mathsf{B}}$, coding the fact that the first successor of the function symbol H is B. We thus obtain $\{\alpha = \mathsf{F}(\mathsf{H}_{1\mathsf{B}}(\alpha), \mathsf{G}(\alpha))$. All the proper subterms of g are now in a simple form, that is, of the form $\mathsf{F}(\vec{\alpha})$, for a function symbol F. We then introduce the new function symbol $\mathsf{F}_{1H_{1\mathsf{B}}2\mathsf{G}}$ which codes the fact that the first successor and the second successor of the function symbol F are $\mathsf{H}_{1\mathsf{H}}$ and G, respectively. Thus, we will obtain the corresponding flat system $\{\alpha = \mathsf{F}_{1\mathsf{H}_{1\mathsf{H}}2\mathsf{G}}(\alpha, \alpha)\}$. We call this new flattening procedure $\mathtt{flat}^*$.

Definition 3.25 Graphs g and h are *strongly bisimilar* if $\mathtt{flat}^*(g) \underline{\leftrightarrow} \mathtt{flat}^*(h)$.

The notion of strong bisimulation corresponds to saying: recursion variables map to recursion variables, while unnamed terms correspond to unnamed terms.

Proposition 3.26 $g \underline{\leftrightarrow}_s h \Longrightarrow g \underline{\leftrightarrow} h$.

Example 3.27

(i) *E.g.,* the terms $\{\alpha = \mathsf{F}(\alpha, \alpha)\}$ and $\{\alpha = \mathsf{F}(\mathsf{F}(\alpha, \alpha), \alpha)\}$ are weakly bisimilar, but not strongly bisimilar, because it entails mapping the recursion variable α to $\mathsf{F}(\alpha, \alpha)$.

(ii) Consider $g_1 \equiv \{\alpha = \mathsf{F}(\alpha, \mathsf{G}(\alpha))\}$ and $g_2 \equiv \{\alpha = \mathsf{F}(\mathsf{F}(\alpha, \mathsf{G}(\alpha)), \mathsf{G}(\alpha))\}$, where $\mathtt{flat}^*(g_1) \equiv \{\alpha = \mathsf{F}_{2G}(\alpha, \alpha)\}$ and $\mathtt{flat}^*(g_2) \equiv \{\alpha = \mathsf{F}_{1F_{2G}2G}(\alpha, \alpha, \alpha)\}$. Then, g_1 and g_2 are not strongly bisimilar even though they have the same tree unwinding, *i.e.,* they are weakly bisimilar.

The notion of functional bisimulation is similarly extended to non-flat terms. Moreover, it is easy to show that the class of strongly bisimilar terms still enjoys lattice properties.

3.4 Another view of term graphs

Instead of starting with a recursion system (term graph) and next introducing the access paths, we can also start from a general space of access paths and 'build' a term graph out of this. The general space of access paths (*i.e.,* Acc) is an object much studied in Mathematical Logic under the name Baire space (when equipped with a topological structure). It consists of all access paths, *i.e.,* finite sequences, $\sigma, \tau, \cdots$, of natural numbers, and can be pictured as the 'universal tree' in Figure 17.

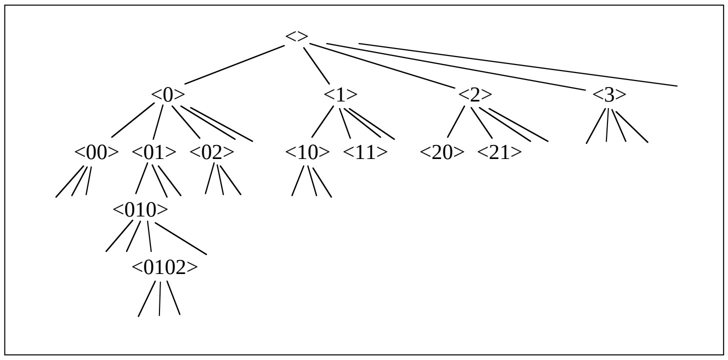


Figure 17.

Definition 3.28 A *tree* T is a subset of Acc satisfying:
(i) $\sigma i \in T \Longrightarrow \sigma \in T$;
(ii) $\sigma i \in T \Longrightarrow \sigma 0, \sigma 1, \cdots \sigma(i-1) \in T$.
(Here σi is short for $\sigma * i$, the concatenation of the sequence σ and the natural number i.)

Definition 3.29 A Σ-*tree* is a tree T with a map $\phi : T \to Fun(\Sigma)$ such that

$$\phi(\sigma) = F^n \Longrightarrow \sigma 0, \cdots, \sigma(n-1) \in T,\ \sigma n \notin T$$

(Here F^n is an n-ary function symbol in the signature Σ.)

Definition 3.30 A Σ-*term graph* G is a Σ-tree T together with a set H of equations between nodes such that
(i) $\sigma = \sigma \in H$
(ii) $\sigma = \tau \in H \Longrightarrow \tau = \sigma \in H$
(iii) $\sigma = \tau \in H, \tau = \rho \in H \Longrightarrow \sigma = \rho \in H$
(iv) $\sigma = \tau \in H \Longrightarrow \sigma\rho = \tau\rho \in H$ for all ρ with $\sigma\rho \in H$
(v) $\sigma = \tau \in H \Longrightarrow \phi(\sigma) = \phi(\tau)$
We will refer to G as (T, H).

Example 3.31 The term graph in Figure 18 is in this representation given by:

$$\begin{aligned} \phi(\langle\rangle) &= \mathsf{F}^2 \\ \phi(\langle 0\rangle) &= \mathsf{G}^1 \\ \phi(\langle 1\rangle) &= \mathsf{H}^1 \end{aligned}$$

H is generated by $\{\langle 0\rangle = \langle 10\rangle, \langle 1\rangle = \langle 00\rangle\}$

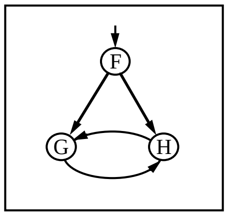


Figure 18.

(The rest of ϕ and H is deducible from the requirements in Definition 3.30. *E.g.,* $\langle 0\rangle = \langle 000\rangle, \langle 1\rangle = \langle 100\rangle \in H$.)

The following facts can be routinely proved.

Proposition 3.32
(i) $G_1 \underline{\leftrightarrow} G_2$ *iff* $G_1 = (T, H_1), G_2 = (T, H_2)$*;*
(ii) $G_1 \underline{\rightarrow} G_2$ *iff* $G_1 = (T, H_1), G_2 = (T, H_2)$ *and* $H_1 \subseteq H_2$*;*
(iii) $(T, H_1) \wedge (T, H_2) = (T, H_1 \cap H_2)$*;*
(iv) $(T, H_1) \vee (T, H_2) = (T, \overline{H_1 \cup H_2})$*, with* $\overline{H_1 \cup H_2}$ *denoting the closure with respect to the rules in Definition 3.30.*
((iii) and (iv) in fact generalize analogously to meet and join of infinite sets $\{(T, H_i) \mid i \in I\}$*.)*

An interesting feature of this representation is that the difference between 'μ-like' graphs and others comes out easily.

Definition 3.33 $G = (T, H)$ is *μ-like* or *has only vertical sharing* if H is generated by a 'basis' $H' \subseteq H$ containing only equations of the form $\sigma = \sigma\tau$ (or $\sigma\tau = \sigma$), called *cyclic* equations.

Example 3.34 Let (T, H) be as in Example 3.31. Then the corresponding C is generated by $\langle 0\rangle = \langle 000\rangle, \langle 1\rangle = \langle 100\rangle$. This yields the term graph in Figure 26(b).

It is easy to determine the minimal μ-like expansion, as follows. Write the graph as a flat system of recursion equations, and then form the corresponding dependency *tree* of the recursion variables, stopping when an earlier variable is reached. Then identify nodes in that tree having the same recursion variables as label. Thus the graph in Figure 18 with recursion system

$$\{\alpha = \mathsf{F}(\beta, \gamma),\ \beta = \mathsf{G}(\gamma),\ \gamma = \mathsf{H}(\beta)\}$$

yields the tree (written as a term)

$$\alpha(\beta(\gamma(\beta)), \gamma(\beta(\gamma)))$$

yielding the node equations $\langle 0\rangle = \langle 000\rangle, \langle 1\rangle = \langle 100\rangle$ found above.

Remark 3.35 Remarkably, the μ-like graphs are not closed under expansion. *E.g.,* the meet of the graphs given by

$$\mu\alpha.\mathsf{F}(\mathsf{F}(\alpha, \alpha), \alpha)$$
$$\mu\alpha.\mathsf{F}(\alpha, \mathsf{F}(\alpha, \alpha))$$

Reflexivity:	$t = t$
Symmetry:	$\dfrac{t_1 = t_2}{t_2 = t_1}$
Transitivity:	$\dfrac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$
Substitution:	$\dfrac{t = s}{t^\sigma = s^\sigma}$ for every substitution σ
Congruence:	$\dfrac{t_1 = s_1 \ \cdots \ t_n = s_n}{\mathsf{F}(t_1, \cdots, t_n) = \mathsf{F}(s_1, \cdots, s_n)}$

Table 2. Equational Logic

is a graph with horizontal sharing, namely

$$\{\alpha \mid \alpha = \mathsf{F}(\beta, \gamma), \beta = \mathsf{F}(\alpha, \gamma), \gamma = \mathsf{F}(\beta, \alpha)\}$$

(See Figure 19.)

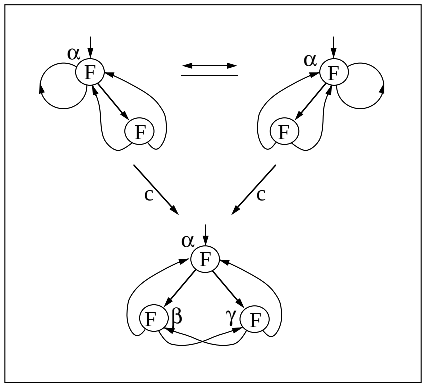


Figure 19.

Remark 3.36 $\mathtt{Lat}(T, H)$ can have at most $2^{\aleph_0}$ elements. This follows at once since there are at most $2^{\aleph_0}$ subsets of H.

For some purposes this view of term graphs as "terms written in homomorphic images of Baire space" is very convenient, as it yields immediately the complete lattice structure of $\underline{\rightarrow}$ and the existence of a minimal μ-like expansion. However, for the purpose of performing actual rewrite steps on term graphs this view seems less suitable than recursion systems.

3.5 Equationally testing for bisimilarity

Reflexivity:	$t = t$
Symmetry:	$\dfrac{t_1 = t_2}{t_2 = t_1}$
Transitivity:	$\dfrac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$
Term decomposition:	$\dfrac{\mathsf{F}(t_1, \cdots, t_n) = \mathsf{F}(s_1, \cdots, s_n)}{t_1 = s_1, \cdots, t_n = s_n}$

Table 3. Syntactic matching

In this section we show how bisimilarity of term graphs can be tested in an equational manner using the proof system of Table 3 ('syntactic matching'). Instead of the congruence rule of Equational Logic (see Table 2) we use (as in some algorithms for syntactic unification of first-order terms [Klo92]) its reverse, the term decomposition rule. This rule says that from $\mathsf{F}(t_1, \cdots, t_n) = \mathsf{F}(s_1, \cdots, s_n)$ we may infer $t_i = s_i \ \ (i = 1, \cdots, n)$. An equation $\mathsf{F}(\vec{t}) = \mathsf{G}(\vec{s})$ with different F,G will be considered a *contradiction.*

Proposition 3.37 *Let $A = \{\alpha_0 = t_0(\vec{\alpha}), \cdots, \alpha_n = t_n(\vec{\alpha})\}$ and $B = \{\beta_0 = s_0(\vec{\beta}), \cdots, \beta_m = s_m(\vec{\beta})\}$ be two term graphs in canonical form. Then $A \underline{\leftrightarrow} B$ iff the equational theory $A \cup B \cup \{\alpha_0 = \beta_0\}$ does not derive a contradiction, using the proof system of Table 3.*

Proof: Let A and B be as described in the hypothesis. We employ the following notation: If $\alpha_p = \mathsf{F}(\cdots, \alpha_q, \cdots)$, where α_q is the i^{th} argument of F_p, we write $\alpha_p \rightarrow_i \alpha_q$. For α_p and $\beta_{p'}$, in different graphs, we write $(\alpha_p, \beta_{p'}) \rightarrow_i (\alpha_q, \beta_{q'})$ as abbreviation for $\alpha_p \rightarrow_i \alpha_q$ and $\beta_{p'} \rightarrow_i \beta_{q'}$. If $\alpha_q = \mathsf{F}_q(\cdots)$, $\beta_{q'} = \mathsf{F}_{q'}(\cdots)$ and $\mathsf{F}_q = \mathsf{F}_{q'}$, we write:

$$(\alpha_p, \beta_p) \rightarrow_i^{\text{match}} (\alpha_q, \beta_{q'})$$

Otherwise, *i.e.,* if $\mathsf{F}_q \neq \mathsf{F}_{q'}$, we write

$$(\alpha_p, \beta_p) \rightarrow_i^{\text{failure}} (\alpha_q, \beta_{q'})$$

Suppose A and B are not bisimilar. Then clearly there must be a sequence (reflecting an unsuccessful attempt to construct a minimal bisimulation starting at relating the roots (α_0, β_0)) as follows:

$$\begin{array}{ll}
(\alpha_0, \beta_0) & \rightarrow_{i_0}^{\text{match}} \\
(\alpha_{f(1)}, \beta_{g(1)}) & \rightarrow_{i_1}^{\text{match}} \\
(\alpha_{f(2)}, \beta_{g(2)}) & \rightarrow_{i_2}^{\text{match}} \\
\vdots & \\
(\alpha_{f(k)}, \beta_{g(k)}) & \rightarrow_{i_k}^{\text{failure}} \\
(\alpha_{f(k+1)}, \beta_{g(k+1)}) &
\end{array}$$

such that the last step is the first failure step. So

$$\alpha_{f(k+1)} = \mathsf{F}(\cdots),\ \beta_{g(k+1)} = \mathsf{G}(\cdots) \quad \text{with } \mathsf{F} \neq \mathsf{G}$$

Obviously this failing attempt corresponds to the derivation of the contradictory equation $\mathsf{F}(\cdots) = \mathsf{G}(\cdots)$, starting from the equational theory:

$$T = A \cup B \cup \{\alpha_0 = \beta_0\}.$$

So we have proved that if there does not exist a bisimulation between A and B, the theory T will derive a "contradiction". The proof of the reverse statement is equally simple and omitted. □

Example 3.38 We want to test the bisimilarity of:

$$A \equiv \{\alpha = \mathsf{F}(\alpha, \beta), \beta = \mathsf{C}\} \quad \text{and} \quad B \equiv \{\gamma = \mathsf{F}(\delta, \epsilon), \delta = \mathsf{F}(\gamma, \epsilon), \epsilon = \mathsf{C}\}.$$

(See the corresponding graphs in Figure 20.)

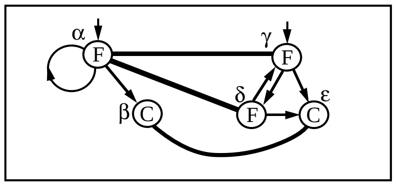


Figure 20.

So consider the theory:

$$T = \{\alpha = \gamma,\ \alpha = \mathsf{F}(\alpha, \beta),\ \beta = \mathsf{C},\ \gamma = \mathsf{F}(\delta, \epsilon),\ \delta = \mathsf{F}(\gamma, \epsilon),\ \epsilon = \mathsf{C}\}.$$

All we can derive from T is: $T' = T \cup \{\mathsf{F}(\alpha, \beta) = \mathsf{F}(\delta, \epsilon),\ \alpha = \gamma,\ \beta = \epsilon,\ \delta = \gamma, \mathsf{F}(\delta, \epsilon) = \mathsf{F}(\gamma, \epsilon)\}$ (apart from equations obtained by reflexivity, symmetry and transitivity). As T' is consistent (*i.e.,* does not contain a contradiction), we conclude $A \underline{\leftrightarrow} B$. Figure 20, with the bisimulation explicitly indicated, confirms this finding. The bisimulation, in the form of equations, is found as a subset of $T' : \{\alpha = \gamma, \alpha = \delta, \beta = \varepsilon\}$.

Remark 3.39 Evidently, the property of bisimilarity for finite graphs is decidable, since the deductive closure T' of T with respect to the proof system of Table 3 (see previous example) is finite for finite T; or, since there are only finitely many relations on a pair of finite graphs.

4. Copying, substitution and flattening

In this section we characterize the fundamental notions of *copying, substitution*, and *flattening* using the simple deductive system of Equational Logic. The notion of copying is also well-known in 'general' graph theory [SS93] under the name 'graph coverings'.

4.1 Copying

Definition 4.1 A *variable substitution* σ is a function from variables to variables. We extend σ to terms and systems of recursion equations, respectively, as follows: $\sigma(\mathsf{F}(t_1, \cdots, t_n)) = \mathsf{F}(\sigma(t_1), \cdots, \sigma(t_n))$ and $\sigma(\{\alpha_0 = t_0, \cdots, \alpha_n = t_n\}) = \{\sigma(\alpha_0) = \sigma(t_0), \cdots, \sigma(\alpha_n) = \sigma(t_n)\}$. We will also write t^σ instead of $\sigma(t)$. A one-to-one variable substitution is also called renaming.

Definition 4.2 $g \longrightarrow_{\mathsf{C}} h$ iff there exists a variable substitution σ such that $h^\sigma = g$, leaving the free variables of h unchanged. We say that h collapses to g or that g copies to h.

E.g.: $g \equiv \{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\alpha)\} \longrightarrow_{\mathsf{C}}$

$$h \equiv \{\alpha = \mathsf{F}(\beta'),\ \beta' = \mathsf{G}(\alpha'),\ \alpha' = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\alpha''),\ \alpha'' = \mathsf{F}(\beta')\}$$

where the variable substitution σ is: $\alpha, \alpha', \alpha''$ are mapped to α, and β, β' are mapped to β (See Figure 21).

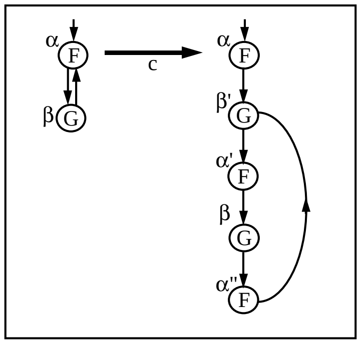


Figure 21.

Proposition 4.3 $g \longrightarrow_{\mathsf{C}} h \Longleftrightarrow h \underline{\rightarrow}_s g$.

Proof: It is easy to check that the variable substitution σ defining the copy operation defines a functional bisimulation from h to g. $\square$

Remark 4.4 In general $h \underline{\rightarrow} g \not\Longleftrightarrow g \longrightarrow_{\mathsf{C}} h$, if h and g are not in flattened form. In fact, there is a functional bisimulation from $g = \{\alpha = \mathsf{F}(\mathsf{F}(\alpha, \alpha), \alpha)\}$ to $h = \{\alpha = \mathsf{F}(\alpha, \alpha)\}$, however, $h \not\longrightarrow_{\mathsf{C}} g$.

Corollary 4.5 $g_1 \underline{\leftrightarrow}_s g_2 \Longrightarrow \exists g_3, g_1 \longrightarrow_{\mathsf{C}} g_3$ *and* $g_2 \longrightarrow_{\mathsf{C}} g_3$.

Proof: Follow from the fact that the strong bisimilarity class is a lattice and from Proposition 4.3. $\square$

Proposition 4.6 $\longrightarrow_{\mathsf{C}}$ *is confluent.*

Proof: Follow from Proposition 4.3 and the above corollary. $\square$

Remark 4.7 In the definition of copying we allow several variable substitutions to occur at once. The question arises whether it suffices to substitute one variable at the time. We will call this restricted version: *sequential copying*, and denote it by $g \longrightarrow_{1\mathsf{c}} g'$. Note that for finite graphs:

$$g \longrightarrow_{1\mathsf{c}} g' \iff g \longrightarrow_{\mathsf{c}} g' \text{ and } \mid g' \mid = \mid g \mid +1$$

where $\mid g \mid$ is the number of nodes of g.

The following example shows that iterated sequential copying is not as powerful as general copying. (*I.e.,* the transitive reflexive closure of $\longrightarrow_{1\mathsf{c}}$ is strictly contained in $\longrightarrow_{\mathsf{c}}$.) Consider $M \equiv \{\alpha = \mathsf{F}(\alpha,\alpha)\} \longrightarrow_{\mathsf{c}} M_1 \equiv \{\alpha = \mathsf{F}(\alpha',\alpha''),\ \alpha' = \mathsf{F}(\alpha',\alpha),\ \alpha'' = \mathsf{F}(\alpha,\alpha'')\}$. We claim that $M \not\twoheadrightarrow_{1\mathsf{c}} M_1$. Suppose otherwise, then for some M_0 we must have $M \longrightarrow_{1\mathsf{c}} M_0 \longrightarrow_{1\mathsf{c}} M_1$. Reasoning backwards, we have four possibilities for M_0:

$$\begin{array}{l}
\{\alpha = \mathsf{F}(\alpha',\alpha), \alpha' = \mathsf{F}(\alpha',\alpha)\} \\
\{\alpha = \mathsf{F}(\alpha',\alpha'), \alpha' = \mathsf{F}(\alpha',\alpha)\} \\
\{\alpha = \mathsf{F}(\alpha,\alpha''), \alpha'' = \mathsf{F}(\alpha,\alpha'')\} \\
\{\alpha = \mathsf{F}(\alpha'',\alpha''), \alpha'' = \mathsf{F}(\alpha,\alpha'')\}
\end{array}$$

However, in none of these cases we have $M_0 \longrightarrow_{1\mathsf{c}} M_1$.

This example is due to Stefan Blom (personal communication), who also proved that sequential copying is sufficient to reach (in the limit) the (infinite) unwinding of a system. Also, for the acyclic case, $\twoheadrightarrow_{1\mathsf{c}}$ and $\longrightarrow_{\mathsf{c}}$ coincide.

4.2 Substitution

Substitution is the operation of substituting the right-hand side of some recursion variable for some occurrences of that variable in the system. E.g. from

$$\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\alpha)\} \qquad (4)$$

we obtain by substitution:

$$\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\mathsf{F}(\beta))\} \qquad (5)$$

and also

$$\{\alpha = \mathsf{F}(\mathsf{G}(\alpha))\} \qquad (6)$$

Such systems are 'not-flat'. We use the notation $\longrightarrow_{\mathsf{s}}$ for the substitution transformation, in fact for the transitive closure. The union of $\longrightarrow_{\mathsf{c}}$ and $\longrightarrow_{\mathsf{s}}$ is $\longrightarrow_{\mathsf{cs}}$. While copying corresponds to strong bisimilarity, substitution corresponds to weak bisimilarity. (Note that (5) and (6) are not strongly bisimilar.)

Proposition 4.8 $\longrightarrow_{\mathsf{s}}$ *and* $\longrightarrow_{\mathsf{cs}}$ *are not confluent.*

Proof: Consider system (4) above, and the transformations (4) $\longrightarrow_{\mathsf{s}}$ (5) and (4) $\longrightarrow_{\mathsf{s}}$ (6). Now (5) and (6) have no common $\longrightarrow_{\mathsf{cs}}$ reduct. In fact, every s or cs reduct of (5) starts with $\alpha = \mathsf{F}(\mathsf{GF})^{2n}()$, while every reduct of (6) starts with $\alpha = (\mathsf{FG})^{2n}()$. While (5) and (6) have the same tree unwinding, they are on the way to that unwinding, irreversibly 'out-of-synch'. □

This non-confluence fact puts a restriction on future developments: we cannot hope to have confluence when combining orthogonal term graph rewrite rules (as in Section 7) with copying and substitution. (However, see Proposition 4.10(ii) below.)

The graphs corresponding to (4) and (6) above are as in Figure 23. Unnamed nodes can be seen as *hidden* or inaccessible nodes. Substitution, as in $\{\alpha = \mathsf{F}(\beta), \beta = \mathsf{G}(\alpha)\}$ to yield $\{\alpha = \mathsf{F}(\mathsf{G}(\alpha))\}$ is communication between the two β's, which subsequently are hidden, nameless.

Remark 4.9 (i) Already 'self-substitution' may be non-confluent; *e.g.,* consider

$$\{\alpha = \mathsf{F}(\alpha,\alpha)\} \longrightarrow_{\mathsf{s}} \{\alpha = \mathsf{F}(\mathsf{F}(\alpha,\alpha),\alpha)\}$$
$$\{\alpha = \mathsf{F}(\alpha,\alpha)\} \longrightarrow_{\mathsf{s}} \{\alpha = \mathsf{F}(\alpha,\mathsf{F}(\alpha,\alpha))\}$$

No further substitutions or copying can make the systems on the right-hand-sides converge again.
(ii) It is easy to see that the presence of cycles is essential to these non-confluence phenomena. Indeed, for acyclic finite systems $\longrightarrow_{\mathsf{cs}}$ is confluent.

4.3 Flattening
Flattening is the operation that takes a non-flat system, and reverts it into a flat system by introducing new recursion variables ('nodes') in a way as general as possible. The effect of flattening on a graph is: *naming unnamed nodes.* In contrast to the two previous operations, the result of flattening is unique (modulo renaming of recursion variables). Notation: $\longrightarrow_{\mathsf{f}}$. For example,

$$\{\alpha = \mathsf{F}(\mathsf{G}(\mathsf{C}),\mathsf{G}(\mathsf{C}))\} \longrightarrow_{\mathsf{f}} \{\alpha = \mathsf{F}(\beta,\gamma), \beta = \mathsf{G}(\delta), \gamma = \mathsf{G}(\epsilon), \delta = \mathsf{C}, \epsilon = \mathsf{C}\}.$$

Note that we do not obtain: $\{\alpha = \mathsf{F}(\beta,\beta), \beta = \mathsf{G}(\delta), \delta = \mathsf{C}\}$. The union of $\longrightarrow_{\mathsf{s}}$ and $\longrightarrow_{\mathsf{f}}$ is $\longrightarrow_{\mathsf{sf}}$, and of $\longrightarrow_{\mathsf{cs}}$ and $\longrightarrow_{\mathsf{f}}$ is $\longrightarrow_{\mathsf{csf}}$.

Substitution and flattening, $\longrightarrow_{\mathsf{s}}$ and $\longrightarrow_{\mathsf{f}}$, are roughly each other's inverse; but not quite, the difference is a copying step $\longrightarrow_{\mathsf{c}}$. This is expressed in (*i*) of the next proposition and in Figure 22, where the dashed arrow has the usual existential meaning.

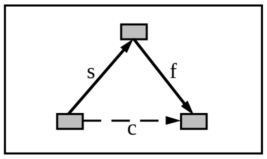


Figure 22.

Proposition 4.10
(i) $\longrightarrow_{\mathsf{s}} \circ \longrightarrow_{\mathsf{f}} \subset \longrightarrow_{\mathsf{c}}$.

(ii) $\longrightarrow_{\mathsf{csf}}$ *is confluent.*

Proof: Routine, using Proposition 4.3.

An example of the strict inclusion in (i) is :

$$g \equiv \{\alpha = \mathsf{F}(\alpha,\alpha)\} \longrightarrow_{\mathsf{c}} g_1 \equiv \{\alpha = \mathsf{F}(\alpha',\alpha''),\ \alpha' = \mathsf{F}(\alpha',\alpha),\ \alpha'' = \mathsf{F}(\alpha,\alpha''\}$$

but $g \not\longrightarrow_{\mathsf{sf}} g_1$. □

Example 4.11 (i)

$$\begin{tikzcd}
\begin{array}{l}\alpha = \mathsf{F}(\beta)\\ \beta = \mathsf{G}(\alpha)\end{array} \arrow[r, "\mathsf{s}"] \arrow[dd, "\mathsf{s}"] & \alpha = \mathsf{F}(\mathsf{G}(\alpha)) \arrow[d, "\mathsf{f}"] \\
& \begin{array}{l}\alpha = \mathsf{F}(\beta)\\ \beta = \mathsf{G}(\alpha)\end{array} \arrow[d, "\mathsf{c}"] \\
\begin{array}{l}\alpha = \mathsf{F}(\beta)\\ \beta = \mathsf{G}(\mathsf{F}(\beta))\end{array} \arrow[r, "\mathsf{f}"] & \begin{array}{l}\alpha = \mathsf{F}(\beta)\\ \beta = \mathsf{G}(\gamma)\\ \gamma = \mathsf{F}(\beta)\end{array}
\end{tikzcd}$$

(ii) by adding a flattening step also the example in Remark 4.9 can be made to commute through a copy step. In fact, both the terms reduce to $\{\alpha = \mathsf{F}(\alpha',\alpha''),\ \alpha' = \mathsf{F}(\alpha,\alpha''),\ \alpha'' = \mathsf{F}(\alpha',\alpha)\}$. Note that in order to find a term h such that $g_1 \longrightarrow_{\mathsf{c}} h$ and $g_2 \longrightarrow_{\mathsf{c}} h$, with $g_1 \underline{\leftrightarrow} g_2$ is enough (by Proposition 4.3) to find the term corresponding to $g_1 \wedge g_2$.

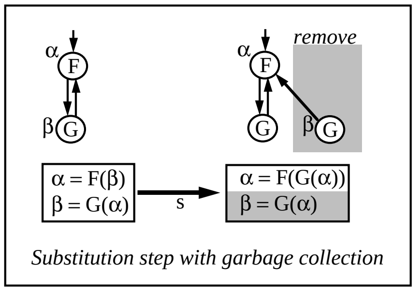


Figure 23.

4.4 Hiding
Nodes that are used only once (that is, with in-degree 1) may be 'hidden'. This means that their name $(\alpha, \beta, \cdots)$ is removed. Notation: $g \longrightarrow_{\mathsf{h}} h$. Hidden or unnamed nodes are 'frozen', and cannot directly be accessed for sharing. Actually, hiding is also the result of substitution. We have the following characterization of $\longrightarrow_{\mathsf{s}}$.

Proposition 4.12 $\longrightarrow_{\mathsf{cs}} = \longrightarrow_{\mathsf{c}} \circ \longrightarrow_{\mathsf{h}}$.

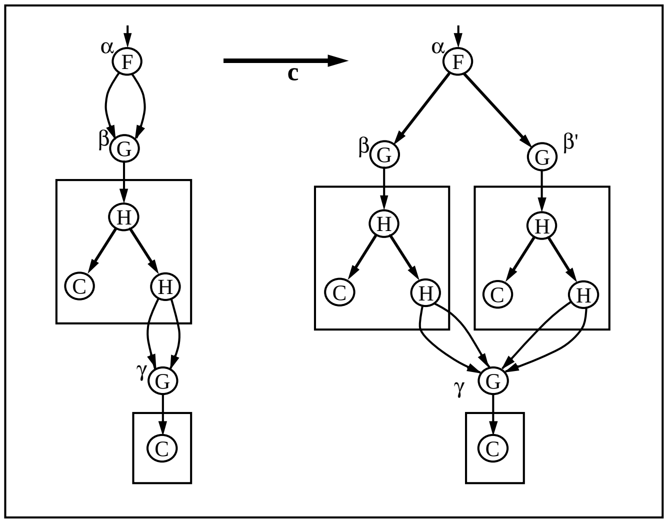
Proof: First prove that $\longrightarrow_{\mathsf{s}} \subseteq \longrightarrow_{\mathsf{c}} \circ \longrightarrow_{\mathsf{h}}$, next prove that in a reduction involving c-steps and h-steps, the h-steps can be postponed to the end. $\square$

Proposition 4.13 $g \longrightarrow_{\mathsf{h}} h \Longleftrightarrow h \longrightarrow_{\mathsf{f}} g$.

Note that hiding comes in naturally once we admit the Equational Logic treatment, hence substitution, and that it has an intuitively plausible interpretation. In fact, it may be seen to be in the spirit of recent proof systems in Linear Logic, where a 'resource-conscious' distinction is made between items (assumptions) that may be used only once and items that may be re-used indefinitely as in classical logic. Barendsen and Smetsers [BS93] introduce explicit notations to introduce 'unique types' for some node graphs, meaning that these are to be used only one, like our hidden nodes.

Essentially, hiding makes it possible to mix terms (tree-like parts) and graphs. For an application exploiting this feature, see the section in the sequel on weakly orthogonality; without hiding there is no natural notion.

Note that copying of a hidden part of a graph requires explicit duplication of that part, as in Figure 24; there is no 'handle' in the hidden part to keep the effect of copying 'local'.



$$\{\ \alpha = \mathsf{F}(\beta,\beta),\quad \beta = \mathsf{G}(\mathsf{H}(\mathsf{C},\mathsf{H}(\gamma,\gamma))),\quad \gamma = \mathsf{G}(\mathsf{C})\} \longrightarrow_{\mathsf{c}} \{\ \alpha = \mathsf{F}(\beta,\beta'),\quad \beta = \mathsf{G}(\mathsf{H}(\mathsf{C},\mathsf{H}(\gamma,\gamma))),\quad \beta' = \mathsf{G}(\mathsf{H}(\mathsf{C},\mathsf{H}(\gamma,\gamma))),\quad \gamma = \mathsf{G}(\mathsf{C})\}$$

Figure 24.

Remark 4.14 The study of recursion equations of course goes back a long way and several notions discussed above occur already *e.g.,* [CKV74, Gue81, dB, Vui73, Vui74, Cou78, CV76, dB80, Cou90]. We will discuss [CKV74] in some detail, also to compare some terminology.

The paper studies 'systems of recursive equations', also called there 'systems of fixed-point equations'. An example is

$$\begin{array}{l} X_1 \Longleftarrow \mathsf{F}(X_1, \mathsf{G}(X_1, X_2)) \\ X_2 \Longleftarrow \mathsf{H}(\mathsf{F}(X_1, X_2)) \\ \text{(principal variable } X_1\text{)} \end{array}$$

where we would write:

$$\begin{array}{l} \alpha = \mathsf{F}(\alpha, \mathsf{G}(\alpha, \beta)) \\ \beta = \mathsf{H}(\mathsf{F}(\alpha, \beta)) \end{array}$$

Our notion of 'flat' is called 'uniform' in [CKV74], where also the procedure of flattening a non-flat system is introduced. Other than in our paper, a general notion of semantics is introduced, using cpo's. Systems are called 'equivalent' if they have the same solution in every model. However, this generality is at once eliminated: systems are equivalent iff they are equivalent with respect to the 'canonical representation', which is nothing else than the semantics of tree unwinding discussed also in this paper.

The paper [CKV74] next discusses procedures to minimize the number of equations in a system, while retaining equivalence. In the course of this it is shown that equivalence of systems (bisimilarity, in our terminology - see Proposition 3.9) is decidable. From a historical point of view, it is interesting that in this proof the notion of bisimilarity (which so to say was lurking around the corner) has not been introduced and used. Instead, the proof is given by constructing the *complement* of a bisimulation, as follows.

Let a flat recursion system $E = \{\alpha_i = t_i(\vec{\alpha}) \mid i = 1, \cdots, n\}$ be given. We want to decide whether α_k and α_l (or rather the subgraphs determined by them) are equivalent. (This is the same problem as deciding whether two recursion systems are equivalent.) Now we construct a sequence of relations $D_0 \subseteq D_1 \subseteq D_2 \subseteq \cdots$, which will be constant eventually, as follows. With F_i we denote the function symbol at the head of t_i, and k_i is the arity of F_i.

$D_0 = \{(\alpha_i, \alpha_j) \mid F_i \neq F_j\}$
$D_{n+1} = D_n \cup \{(\alpha_i, \alpha_j) \mid F_i \equiv F_j \text{ and for some } m \text{ with } 1 \leq m \leq k_i, (\alpha_{i_m}, \alpha_{j_m}) \in D_n\}$
Here $\alpha_i \rightarrow_m \alpha_{i_m}$ *etc.*,

It is clear that for some N, $D_N = D_{N+1}$. Now [CKV74] states: α_i is equivalent with α_j iff $(\alpha_i, \alpha_j) \notin D_N$. In fact the complement of D_N, *i.e.,* $(\mathtt{NODES}(E) \times \mathtt{NODES}(E)) - D_N$, is just the maximal 'auto-bisimulation' of E.

Note however that the work cited above does not relate systems of equations to term graph rewriting as we do in this paper. Their notion of equivalence corresponds to our notion of bisimilarity but not to the strong version of it. *E.g.,* the systems $X \Longleftarrow \mathsf{F}(X, \mathsf{G}(X))$ and $X \Longleftarrow \mathsf{F}(\mathsf{F}(Y, \mathsf{G}(Y)), \mathsf{G}(Y))$ are equivalent in [CKV74], whereas we do not consider them to be strongly bisimilar. Moreover, differently from [CKV74], in this paper we devise a computation rule, *i.e.,* copying, which yields equivalent systems (in the strong sense) and still maintains confluence.

4.5 Acyclic substitution

We next define a notion of substitution which avoids the non-confluence trap. The new substitution is called *acyclic substitution*, written as $\longrightarrow_{\mathsf{as}}$, and consists in defining an order on the nodes in a graph as in Figure 25, and then allowing substitution upwards only. More

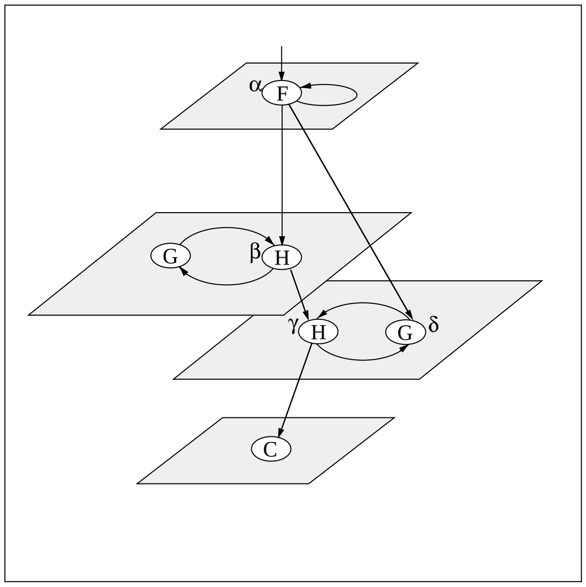


Figure 25.

precisely: call two nodes *cyclically equivalent* if they are lying on a common cycle. A *plane* is a cyclic equivalence class. If there is a path from node s to node t, and s, t are not in the same plane, we define $s > t$. Now suppose $\alpha > \beta$. Then

$$\{\cdots, \alpha = t(\beta), \cdots, \beta = s, \cdots\} \longrightarrow_{\mathsf{as}} \{\cdots, \alpha = t(s), \cdots, \beta = s, \cdots\}.$$

Here in $t(\beta)$ just one occurrence of β is displayed and replaced by s. So in Figure 25, displaying the system

$$\{\alpha = \mathsf{F}(\beta, \delta, \alpha), \beta = \mathsf{H}(\mathsf{G}(\beta), \gamma), \gamma = \mathsf{H}(\mathsf{C}, \delta), \delta = \mathsf{G}(\gamma)\}$$

the only $\longrightarrow_{\mathsf{as}}$ -steps are from δ in α, from β in α, from γ in β.

Proposition 4.15 $\longrightarrow_{\mathsf{as}}$ *is confluent.*

Proof: We first show that $\longrightarrow_{\mathsf{as}}$ is weakly-confluent. Consider two diverging acyclic substitutions, the first for β in α, the second for δ in γ. So we have the system equations as follows:

$$\begin{array}{rcl} \alpha & = & \cdots\beta\cdots \\ \beta & = & ---\\ \gamma & = & ***\delta*** \\ \delta & = & === \end{array}$$

Here α, β, γ, δ need not be all different. Clearly, $\alpha \neq \beta$ and $\gamma \neq \delta$ since the substitutions are acyclic. Now distinguish:

1: Case 1. α, β and γ, δ have empty intersection. Then the two substitutions are trivially commuting.

2: Case 2. The two sets as in case 1 have one element in common.

2.1: Case 2.1. $\alpha = \gamma$. Then the situation is

$$\begin{array}{lcl} \alpha & = & \cdots\beta\cdots\delta\cdots \\ \beta & = & - - - \\ \delta & = & === \end{array}$$

and the commuting property of the two substitutions is again trivial.

2.2: Case 2.2. $\alpha = \delta$. Then we have

$$\begin{array}{lcl} \gamma & = & ***\delta *** \\ \delta & = & \cdots\beta\cdots \\ \beta & = & - - - \end{array}$$

Again the substitutions commute. However, in this case we need to perform two substitutions step, one for β in δ and one for β in γ.

2.3: Case 2.3. $\beta = \delta$. Easy.

3: Case 3. $\alpha, \beta = \gamma, \delta$

3.1: Case 3.1. $\alpha = \gamma$, $\beta = \delta$. We have

$$\begin{array}{lcl} \alpha & = & \cdots\beta\cdots\beta\cdots \\ \beta & = & - - - \end{array}$$

and commutation is trivial.

3.2: Case 3.2. $\alpha = \delta, \beta = \gamma$. This case is ruled out by acyclicity of the substitutions.

From the above case analysis it is easy to see that $\longrightarrow_{\mathsf{as}}$ satisfies the parallel move lemma, and thus is confluent. □

As before, the notion of $\longrightarrow_{\mathsf{as}}$ is not primitive and can be analyzed in terms of copying and *acyclic hiding* ($\longrightarrow_{\mathsf{ah}}$). This is hiding of a node not on a cycle.

Proposition 4.16 $\longrightarrow_{\mathsf{as}} \subset \longrightarrow_{\mathsf{c}} \circ \longrightarrow_{\mathsf{ah}}$

As an example of the strict inclusion in the above proposition consider:

$$\begin{array}{l} M \equiv \{\alpha = \mathsf{F}\beta, \beta = \mathsf{G}\alpha\} \longrightarrow_{\mathsf{c}} \{\alpha = \mathsf{F}\beta, \beta = \mathsf{G}\alpha', \alpha' = \mathsf{F}\beta', \beta' = \mathsf{G}\alpha'\} \longrightarrow_{\mathsf{ah}} \\ \qquad\qquad\qquad\qquad\qquad \{\alpha = \mathsf{FG}\alpha', \alpha' = \mathsf{F}\beta', \beta' = \mathsf{G}\alpha'\} \end{array}$$

but $M \not\longrightarrow_{\mathsf{as}} M_1$.

Theorem 4.17 $\longrightarrow_{\mathsf{c}} \cup \longrightarrow_{\mathsf{ah}}$ *is confluent.*

Remark 4.18 Copying, however, does not commute with acyclic hiding.

$$\begin{tikzcd}
\begin{array}{l} \alpha = \mathsf{F}(\beta,\beta) \\ \beta = \mathsf{G}(\delta) \\ \delta = 0 \end{array} \arrow[rr, "\mathsf{ah}"] \arrow[d, "\mathsf{c}"] & & \begin{array}{l} \alpha = \mathsf{F}(\beta,\beta) \\ \beta = \mathsf{G}(0) \end{array} \arrow[d, "\mathsf{c}"] \\
\begin{array}{l} \alpha = \mathsf{F}(\beta',\beta) \\ \beta = \mathsf{G}(\delta) \\ \beta' = \mathsf{G}(\delta) \\ \delta = 0 \end{array} \arrow[r, "\mathsf{c}"'] & {} \arrow[r, "\mathsf{ah}"'] & \begin{array}{l} \alpha = \mathsf{F}(\beta',\beta) \\ \beta = \mathsf{G}(0) \\ \beta' = \mathsf{G}(0) \end{array}
\end{tikzcd}$$

Before doing the hiding an extra copy step is required.

Remark 4.19 Another easy way of avoiding the out-of-synch phenomenon is by performing a *parallel substitution*, which consists in substituting at once for all the recursion variables. Notation: $\longrightarrow_{\mathsf{ps}}$.

$$\{\alpha_1 = t_1, \cdots, \alpha_n = t_n\} \longrightarrow_{\mathsf{ps}} \{\alpha_1 = t_1[\vec{\alpha_n}/\vec{t_n}], \cdots, \alpha_n = t_n[\vec{\alpha_n}/\vec{t_n}]\}$$

For example, we have

$$\{\alpha = \mathsf{F}(\beta), \beta = \mathsf{G}(\gamma), \gamma = \mathsf{H}(\alpha)\} \longrightarrow_{\mathsf{ps}} \{\alpha = \mathsf{F}(\mathsf{G}(\gamma)), \beta = \mathsf{G}(\mathsf{H}(\alpha)), \gamma = \mathsf{H}(\mathsf{F}(\beta))\}$$

We expect that $\longrightarrow_{\mathsf{ps}} \cup \longrightarrow_{\mathsf{c}}$ is confluent, but refrain from proving this as this notion of substitution is less interesting to us (we always have references to the original nodes, which seems not to be a proper form of unwinding).

5. Translations between μ-terms and recursion systems

5.1 *Translation of a μ-term into a recursion system*

We will transform a μ-term in a number of steps into a recursion system. During the procedure, named γ, we have a recursion system in which also μ-terms may appear on the right-hand side of the equations. Let M be a μ-term. Suppose M has been α-converted (renamed) such that all variables bound by μ are distinct. Moreover, rename those variables in greek letters $\alpha, \beta, \cdots$.

(i) If M does not start with μ, or if M is β, we write $\alpha = M$;
(ii) if M is $\mu\beta.N$, we write $\beta = N$;
(iii) let one of the right-hand sides of the equations contain a subterm $\mu\delta.P$. Then add the equation $\delta = P$ to the system, and replace $\mu\delta.P$ by δ;
(iv) if no μ appears in the system, remove equations of the form $\alpha = \beta$ after substituting β for all (other) occurrences of α;
(v) replace an equation $\alpha = \alpha$ by $\alpha = \bullet$.

Remark 5.1 The translation γ is not injective. In fact, precisely the μ-terms that are provably equal by means of the following identities (all provable from EL_μ) are identified by γ:

$$\begin{array}{l} \mu\alpha.t = t \text{ if } \alpha \text{ does not occur free in } t \\ \mu\alpha.\mu\beta.t = \mu\beta.\mu\alpha.t \\ \mu\alpha.\mu\beta.t(\alpha,\beta) = \mu\alpha.t(\alpha,\alpha). \end{array}$$

Example 5.2

(i) $\mu\alpha.\mu\beta.\mathsf{F}(\alpha,\beta) \longrightarrow$

$\alpha = \mu\beta.\mathsf{F}(\alpha,\beta) \longrightarrow$

$\alpha = \beta,\ \beta = \mathsf{F}(\alpha,\beta) \longrightarrow$

$\beta = \mathsf{F}(\beta,\beta)$

(ii) $\mu\alpha.\mathsf{F}(\mathsf{G}(\mathsf{H}(\alpha))) \longrightarrow$

$\alpha = \mathsf{F}(\mathsf{G}(\mathsf{H}(\alpha)))$

The second example indicates that the translation may result in a non-flat system.

We claim that γ as defined here, yields the same graph as γ in Definition 2.7. We omit the tedious routine proof of this claim. Also: γ is non-deterministic, but yields a unique result.

5.2 Translation of a recursion system into a μ-term

We will now define a translation:

$$\mu : \mathrm{Graph}(\Sigma) \to \mathrm{Term}(\Sigma_\mu)$$

We will do so by using an auxiliary function μ and a notion of 'environments':

$$mu : \mathrm{Term}(\Sigma) \times \mathrm{Env}(\Sigma) \to \mathrm{Term}(\Sigma_\mu)$$

Thus:

$$\mu(\{\alpha|E\}) = mu^E(\alpha)$$

where $E \in \mathrm{Env}(\Sigma)$ is a set of recursion equations. (So E is a system of recursion equations as before, but without indication of a root equation.)

$$\begin{array}{lcl} mu^E(\alpha) & = & \alpha \quad \text{if } \alpha \text{ does not occur in the left-hand side of any equation in } E \\ mu^{\{\alpha=t\}\cup E}(\alpha) & = & \mu\alpha.mu^E(t) \\ mu^E(\mathsf{F}^n(t_1,\cdots,t_n)) & = & \mathsf{F}^n(mu^E(t_1),\cdots,mu^E(t_n)) \end{array}$$

Now define for a system of recursion equations $\{\alpha \mid E\}$:

$$\mu(\{\alpha \mid E\}) = mu^E(\alpha)$$

Remark 5.3 Let E be $\{\alpha_0 \mid \alpha_0 = t_0(\vec{\alpha}),\cdots,\alpha_n = t_n(\vec{\alpha})\}$. Then:

$$\mu(\{\alpha_0 \mid E\}) = \mu\alpha_0.t_0(\alpha_0,\mu(\{\alpha_1 \mid E-\alpha_0\}),\cdots,\mu(\{\alpha_n \mid E-\alpha_0\})\})$$

where $E-\alpha_0$ is the set of equations $\{\alpha_1 = t_1(\vec{\alpha}),\cdots,\alpha_n = t_n(\vec{\alpha})\}$.

Example 5.4 $\{\alpha \mid E\} = \{\alpha \mid \alpha = \mathsf{F}(\beta,\eta),\ \beta = \mathsf{G}(\eta),\ \eta = \mathsf{H}(\beta)\}$. Then:

$$\begin{array}{l} \mu\alpha(\{\alpha \mid E\} = mu^E(\alpha) = \\ \mu\alpha.mu^{\{\beta=\mathsf{G}(\eta),\ \eta=\mathsf{H}(\beta)\}}(\mathsf{F}(\beta,\eta)) = \\ \mu\alpha.\mathsf{F}(mu^{\{\beta=\mathsf{G}(\eta),\ \eta=\mathsf{H}(\beta)\}}(\beta),mu^{\{\beta=\mathsf{G}(\eta),\ \eta=\mathsf{H}(\beta)\}}(\eta)) = \\ \mu\alpha.\mathsf{F}(\mu\beta.mu^{\{\eta=\mathsf{H}(\beta)\}}(\mathsf{G}(\eta)),\mu\eta.mu^{\{\beta=\mathsf{G}(\eta)\}}(\mathsf{H}(\beta))) = \\ \mu\alpha.\mathsf{F}(\mu\beta.\mathsf{G}(mu^{\{\eta=\mathsf{H}(\beta)\}}(\eta)),\mu\eta.\mathsf{H}(mu^{\{\beta=\mathsf{G}(\eta)\}}(\beta))) = \\ \mu\alpha.\mathsf{F}(\mu\beta.\mathsf{G}(\mu\eta.mu^{\emptyset}(\mathsf{H}(\beta))),\mu\eta.\mathsf{H}(\mu\beta.mu^{\emptyset}(\mathsf{G}(\eta)))) = \\ \mu\alpha.\mathsf{F}(\mu\beta.\mathsf{G}(\mu\eta.\mathsf{H}(\beta)),\mu\eta.\mathsf{H}(\mu\beta.\mathsf{G}(\eta))). \end{array}$$

Note that the binder $\mu\alpha$ turns out to be superfluous, and likewise the first $\mu\eta$ and the second $\mu\beta$. Removing these we obtain

$$\mathsf{F}(\mu\beta.\mathsf{G}(\mathsf{H}(\beta)), \mu\eta.\mathsf{H}(\mathsf{G}(\eta))).$$

Remark 5.5 It is interesting to compare the translation μ with the following non-deterministic translation procedure μ', which roughly seems to be the procedure that is the inverse of γ in the previous section:
(i) Replace the root equation $\alpha = M$ by $\mu\alpha.M$;
(ii) let the system contain an equation $\beta = N$. Then: omit this equation, and replace each occurrence of β by $\mu\beta.N$;
(iii) repeat until no more equations are present.
(Optimization: if in (i), M does not contain α (directly or indirectly, via other equations), then $\mu\alpha.M = M$ and we can replace just by M. Likewise for (ii).)
As before the translation is performed in a number of steps, during which we have a hybrid μ-term.

Note that the result of the above procedure is not unique. In fact, let M be $\{\alpha \mid E\}$ as in Example 5.4. Then applying μ' on M may yield both

$$\mathsf{F}(\mathsf{G}(\mu\eta.\mathsf{H}(\mathsf{G}(\eta))), \mu\eta.(\mathsf{H}(\mathsf{G}(\eta))))$$

or

$$\mathsf{F}(\mu\beta.\mathsf{G}(\mathsf{H}(\beta)), \mathsf{H}(\mu\beta.\mathsf{G}(\mathsf{H}(\beta)))).$$

However, we do not obtain the minimal term containing vertical sharing only, which was found in Example 5.4. The above terms and the minimal one are displayed in Figure 26.

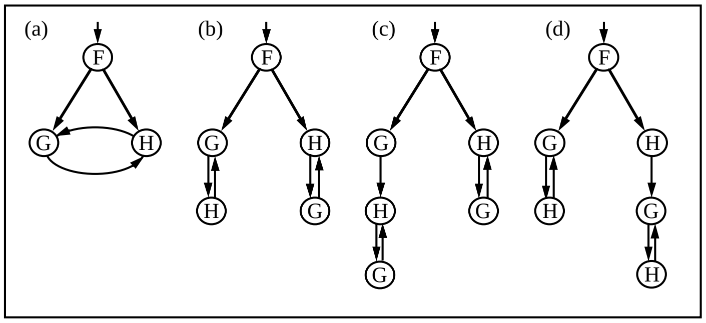


Figure 26.

The following proposition states that the translation μ indeed is 'best possible': it only removes the horizontal sharing, but preserves all the vertical sharing.

Proposition 5.6 *Let M a term graph. Then:*
(i) $M \longrightarrow_{\mathsf{c}} \gamma(\mu(M))$;
(ii) *$\gamma(\mu(M))$ has only vertical sharing;*
(iii) *if $M \longrightarrow_{\mathsf{c}} M'$ and M' has only vertical sharing, then $\gamma(\mu(M)) \longrightarrow_{\mathsf{c}} M'$.*

Reflexivity:	$t = t$	
Symmetry:	$\dfrac{t_1 = t_2}{t_2 = t_1}$	
Transitivity:	$\dfrac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$	
Substitution:	$\dfrac{t_1 = s_1 \quad t = t'}{t_1[x := t] = s_1[x := t']}$	
Unwinding:	$\mu\alpha.t(\alpha) = t(\mu\alpha.t(\alpha))$	
Renaming:	$\mu\alpha.t(\alpha) = \mu\beta.t(\beta)$	
Folding:	$\dfrac{t_1 = t(t_1)}{t_1 = \mu\alpha.t(\alpha)}$	α guarded in $t(\alpha)$

Table 4. EL_μ

Proof: Sketch: We use the representation of a term graph as $G = (T, H)$ of Section 3.4. Suppose the term graph G exhibits a cyclic node equation $\sigma = \sigma\tau$. For convenience, suppose the recursion system corresponding to G is flat. The cyclic equation is visible as a cyclic dependence of the recursion variables. An inspection of the translation procedure μ readily shows that the cyclic dependence stays preserved. Hence the result of the translation yields the 'minimally horizontally-unshared' term graph. □

6. A COMPLETE PROOF SYSTEM FOR μ-TERMS

As an application of the theory for term graphs discussed in Sections 3 and 4, and the translations given in the previous section, we present a simple proof of completeness of the proof system EL$_\mu$ shown in Table 4. Completeness is with respect to the semantics of infinite unwinding. The theorem is undoubtedly well-known to many people, also because it is analogous to the completeness theorem in Milner [Mil84] for μ-terms in process algebra ('regular behaviors'). At present we do not know references to proofs in the literature (actually, this question for references including the proof system in Table 4 was mentioned to us in correspondence by T. Nipkow); anyway we include the proof below because it shows the convenience of reasoning with the concepts of bisimulation and copying.

Definition 6.1 A recursion variable α is *guarded in* $t(\alpha)$ if α is preceded by a function symbol other than μ (equivalently, if $t(\alpha)$ contains a function symbol other than μ).

Example 6.2 The following derivation shows that $EL_\mu \vdash \mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon) = \mu\beta.\mu\alpha.\mathsf{F}(\alpha,\beta)$.

$$\begin{array}{ll}
\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon) = \mathsf{F}(\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon),\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon)) & \text{by unwinding}\\
\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon) = \mu\alpha.\mathsf{F}(\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon),\alpha) & \text{by folding}\\
\mu\varepsilon.\mathsf{F}(\varepsilon,\varepsilon) = \mu\beta\alpha.\mathsf{F}(\beta,\alpha) & \text{by folding}
\end{array}$$

Remark 6.3 Note that the proviso in the folding rule is necessary; without it we could derive:

$$\dfrac{\dfrac{M = M}{M = \mu\alpha.\alpha} \quad \dfrac{\dfrac{N = N}{N = \mu\alpha.\alpha}}{\mu\alpha.\alpha = N}}{M = N}$$

The addition in Definition 6.1 "other than μ" is necessary too; otherwise we would have for M not containing α:

$$\dfrac{\dfrac{\dfrac{\mu\alpha.M = M}{M = \mu\alpha.M}}{M = \mu\beta.\mu\alpha.\beta}}{M = \mu\beta.\beta}$$

Let Term(Σ_μ) be the set of μ-terms over the signature Σ, let Graph(Σ) be the set of flat recursion systems over Σ, and let Term$^\infty(\Sigma)$ be the set of possibly infinite terms (trees) over Σ. As defined before, we have a semantic mapping $[\![\ ]\!]$, denoting infinite unwinding:

$$\begin{array}{rcl}
[\![\,]\!] : \ \text{Term}(\Sigma_\mu) & \rightarrow & \text{Term}^\infty(\Sigma)\\
[\![\,]\!] : \ \text{Graph}(\Sigma) & \rightarrow & \text{Term}^\infty(\Sigma)
\end{array}$$

(Actually, we should use $[\![\,]\!]_1$ and $[\![\,]\!]_2$, but our ambiguous notation will not cause confusion.) Let $\gamma : \text{Term}(\Sigma_\mu) \rightarrow \text{Graph}(\Sigma)$ and $\mu : \text{Graph}(\Sigma) \rightarrow \text{Term}(\Sigma_\mu)$ be the translations as in the previous section.

Proposition 6.4 *Let $M \in Term(\Sigma_\mu)$, $E \in Graph(\Sigma)$. Then:*

$$(i)[\![E]\!] = [\![\mu(E)]\!] \text{ and } (ii)[\![M]\!] = [\![\gamma(M)]\!].$$

That is, γ and μ are sound with respect to $[\![\,]\!]$.

Proof: (i): easy induction on the number of recursive equations.
(ii): by structural induction on M. □

Definition 6.5 Let $M_0 \in Term(\Sigma_\mu)$. Then:

$$M_0 \models E \ (M_0 \textit{ solves } E)$$

if $E = \{\alpha_0 = t_0(\vec{\alpha}), \cdots, \alpha_n = t_n(\vec{\alpha})\}$ and there are $M_1, \cdots, M_n \in \text{Term}(\Sigma_\mu)$ such that

$$EL_\mu \vdash M_i = t_i(\vec{M}) \text{ for all } i = 0, \cdots, n.$$

Example 6.6 (i) Let E be $\{\alpha = \mathsf{F}(\alpha)\}$, we have $\mu\alpha.\mathsf{F}(\alpha) \models E$, because $\mu\alpha.\mathsf{F}(\alpha) = \mathsf{F}(\mu\alpha.\mathsf{F}(\alpha))$ by applying the unwinding rule.
(ii) If E is $\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\alpha)\}$ then $\mu\gamma.\mathsf{F}(\mathsf{G}(\gamma)) \models E$, because

$$(1)\ EL_\mu \vdash \mu\gamma.\mathsf{F}(\mathsf{G}(\gamma)) = \mathsf{F}(\mu\delta.\mathsf{G}(\mathsf{F}(\delta)))$$

by the application of the unwinding, folding and transitive rule.

$$(2)\ EL_\mu \vdash \mu\delta.\mathsf{G}(\mathsf{F}(\delta)) = \mathsf{G}(\mu\gamma.\mathsf{F}(\mathsf{G}(\gamma))).$$

Proposition 6.7 *Let $M \in Term(\Sigma_\mu)$, $E \in Graph(\Sigma)$. Then:*

$$M \models E \Longrightarrow EL_\mu \vdash M = \mu(E).$$

Proof: We must prove a stronger statement in order to let the following induction argument go through. To that end, let $N, M_1, \cdots, M_k \in \mathrm{Term}(\Sigma_\mu)$. We define $N \models E$ for systems $E = \{\alpha_0 = t_0(\vec{\alpha}, \vec{M}), \cdots, \alpha_n = t_n(\vec{\alpha}, \vec{M})\}$ as before, but with the difference that the right-hand sides of the equations in E now are allowed to contain occurrences of $\vec{M} = M_1, \cdots, M_k$ as indicated. Notation: $E(\vec{\beta})[\vec{\beta} := \vec{M}]$ denotes the substitution of $M_1, \cdots, M_k$ for $\beta_1, \cdots, \beta_k$, respectively.

We aim to prove:
Let $E(\vec{\beta})$ be the system of recursion equations $\{\alpha_0 = t_0(\vec{\alpha}, \vec{\beta}), \cdots, \alpha_n = t_n(\vec{\alpha}, \vec{\beta})\}$. (Here $\vec{\beta} = \beta_1, \cdots, \beta_k$ are free recursion variables.) Then:

$$M_0 \models E(\vec{\beta})[\vec{\beta} := \vec{N}] \Longrightarrow EL_\mu \vdash M_0 = \mu(E(\vec{\beta}))[\vec{\beta} := \vec{N}].$$

(Henceforth we will write just $\vdash$ for $EL_\mu \vdash$.)
The proof is by induction on the number of equations in E.

1: Base case: easy.

2: Induction step: suppose proved for n. Now consider M_0, $E(\vec{\beta})$, $\vec{N}$ as in the statement to prove. Let $M_1, \cdots, M_n$ be the auxiliary solutions, *i.e.*,

$$\vdash M_i = t_i(M_0, M_1, \cdots, M_n, \vec{N})\ \ (i = 0, \cdots, n)$$

So $M_i \models \{\alpha_i \mid E - \alpha_0\}[\alpha_0, \vec{\beta} := M_0, \vec{N}]$ $(i = 1, \cdots, n)$. By induction hypothesis : $\vdash M_i = \mu(\{\alpha_i \mid E - \alpha_0\})[\alpha_0, \vec{\beta} := M_0, \vec{N}]$ $(i = 1, \cdots, n)$. We know: $\vdash M_0 = t_0(M_0, M_1, \cdots, M_n, \vec{N})$. So:

$$\vdash M_0 = t_0(M_0, \mu(\{\alpha_1 \mid E{-}\alpha_0\})[\alpha_0 := M_0], \cdots, \mu(\{\alpha_n \mid E{-}\alpha_0\})[\alpha_0 := M_0], \vec{\beta})[\vec{\beta} := \vec{N}]$$

But then by folding:

$$\vdash M_0 = \mu\alpha_0.t_0(\alpha_0, \mu(\{\alpha_1 \mid E - \alpha_0\}), \cdots, \mu(\{\alpha_n \mid E - \alpha_0\}), \vec{\beta})[\vec{\beta} := \vec{N}]$$

By Remark 5.3:

$$\vdash M_0 = \mu(\{\alpha_0 \mid E(\vec{\beta})\})[\vec{\beta} := \vec{N}]$$

which ends the proof.

$\square$

Proposition 6.8 *Let $M \in Term(\Sigma_\mu)$, $E \in Graph(\Sigma)$. Then:*

$$M \models E \text{ and } E \longrightarrow_{\mathsf{c}} E' \Longrightarrow M \models E'.$$

Proof: To avoid cumbersome notation we consider an example. Let E be $\{\alpha = \mathsf{F}(\alpha, \beta),\ \beta = \mathsf{G}(\alpha)\}$. Suppose $M \models E$; so there is an N such that $EL_\mu \vdash M = \mathsf{F}(M, N)$ and $EL_\mu \vdash N = \mathsf{G}(M)$. Now suppose $E \longrightarrow_{\mathsf{c}} E'$ via addition of equations $\alpha' = \alpha, \alpha'' = \alpha, \beta' = \beta$, next deriving a canonical system E'. Then it is clear that $M \models E'$, by substituting for $\alpha, \alpha', \alpha''$ respectively β, β' the μ-terms M, M, M respectively N, N. $\square$

Proposition 6.9 *For $M \in Term(\Sigma_\mu)$: $M \models \texttt{flat}(\gamma(M))$.*

Proof: The translation procedure γ yields starting from M, a sequence of hybrid recursion systems (*i.e.,* where right-hand sides may contain μ-terms). The relation $\models$ is extended to such systems as in the proof of Proposition 6.7. Let this sequence be $E_0, E_1, \cdots, E_k = \gamma(M)$. We prove $M \models E_i\ \ (i = 0, \cdots, k)$.

1: Base case: If E_0 was obtained by applying clause (i) of the definition of γ, then trivially $M \models \{\alpha = M\}$. If E_0 was obtained by clause (ii), then $M \equiv \mu\beta.N \models \{\beta = N\}$ by applying the unwinding axiom of EL_μ.

2: Induction step: Suppose $M \models E_m$. Let E_{m+1} be obtained by lifting out an occurrence of $\mu\delta.\mathsf{P}(\delta, \vec{\alpha})$ from a right-hand side, replacing it by δ, and adding $\delta = \mathsf{P}(\delta, \vec{\alpha})$ to the system. Now it is clear that adding $\mu\delta.\mathsf{P}(\delta, \underline{\vec{\alpha}})$ as a solution for δ does the job. Here $\underline{\vec{\alpha}}$ denotes the tuple of solutions for $\vec{\alpha}$ that are already present.

$\square$

Theorem 6.10 *Let $M_1, M_2 \in Term(\Sigma_\mu)$. Then:*

$$[\![M_1]\!] = [\![M_2]\!] \Longleftrightarrow EL_\mu \vdash M_1 = M_2.$$

Proof: Soundness ($\Longleftarrow$): clear. Completeness ($\Longrightarrow$): By soundness of γ we have:

$$[\![M_1]\!] = [\![M_2]\!] = [\![\gamma(M_1)]\!] = [\![\gamma(M_2)]\!].$$

Since flattening does not affect the basic structure of a term, we have that $[\![\texttt{flat}(\gamma(M_1))]\!] = [\![\texttt{flat}(\gamma(M_2))]\!]$. This means that $\texttt{flat}(\gamma(M_1)) \underline{\leftrightarrow}_s \texttt{flat}(\gamma(M_2))$, and by Corollary 4.5 there exists an E such that $\texttt{flat}(\gamma(M_i)) \longrightarrow_{\mathsf{c}} E$ $(i = 1, 2)$. By Proposition 6.9 we have $M_i \models \texttt{flat}(\gamma(M_i)), i = 1, 2$. Hence, by Proposition 6.8, $M_i \models E, i = 1, 2$. By Proposition 6.7: $EL_\mu \vdash M_i = \mu(E)$. So $EL_\mu \vdash M_1 = M_2$. $\square$

Example 6.11 We want to show that the following two terms M_1 and M_2 are provably equal:

$$\mu\alpha.\mathsf{F}(\alpha, \mathsf{F}(\alpha, \alpha)) \text{ and } \mu\alpha.\mathsf{F}(\mathsf{F}(\alpha, \alpha), \alpha).$$

Following the steps of the proof we derive that we need to show them both equal to

$$\mu\alpha.\mathsf{F}(\mu\beta.\mathsf{F}(\alpha,\mathsf{F}(\beta,\alpha)),\mu\beta.\mathsf{F}(\mathsf{F}(\alpha,\beta),\alpha)).$$

In fact

$\vdash M_1 = \mathsf{F}(M_1,\mathsf{F}(M_1,M_1))$	unwinding rule
$\vdash M_1 = \mu\beta.\mathsf{F}(M_1,\mathsf{F}(\beta,M_1))$	folding rule
$\vdash \mathsf{F}(M_1,M_1) = \mathsf{F}(\mathsf{F}(M_1,\mathsf{F}(M_1,M_1)),M_1)$	unwinding rule
$\vdash \mathsf{F}(M_1,M_1) = \mu\beta.\mathsf{F}(\mathsf{F}(M_1,\beta),M_1)$	folding rule
$\vdash M_1 = \mathsf{F}(\mu\beta.\mathsf{F}(M_1,\mathsf{F}(\beta,M_1)),\mu\beta.\mathsf{F}(\mathsf{F}(M_1,\beta),M_1))$	transitivity rule
$\vdash M_1 = \mu\alpha.\mathsf{F}(\mu\beta.\mathsf{F}(\alpha,\mathsf{F}(\beta,\alpha)),\mu\beta.\mathsf{F}(\mathsf{F}(\alpha,\beta),\alpha))$	folding rule

Similarly for M_2.

7. Orthogonal term graph rewriting with copying

Analogous to term graphs, graph rewrite rules are also expressed in equational format. For example, the cyclic Y-rule depicted in Figure 1, is expressed as:

$$\{\alpha = \mathsf{Ap}(\beta,\gamma),\ \beta = \mathsf{Y}\} \longrightarrow \{\alpha = \mathsf{Ap}(\gamma,\alpha)\}$$

Definition 7.1 Let l and r be term graphs with the same root. Then: $l \to r$ is a *term graph rule.*

As is customary in TRSs two conditions are imposed on rules. Namely, (1) the left-hand side cannot be of the form $\{\alpha = \beta\}$, (2) the free variables occurring in the right-hand side are a subset of those occurring in the left-hand side. However, rules are not restricted to flat systems only. For example, $\{\alpha = \mathsf{F}(\mathsf{G}(0))\} \longrightarrow \{\alpha = 0\}$ is a legitimate rule.

Definition 7.2 Let $\tau : l \to r$. The rule τ is said to be *left-linear* if for all variables α occurring in l, $\mathtt{Acc}(\alpha)$ is a singleton.

In other words, the rules $\{\alpha = \mathsf{F}(\alpha)\} \longrightarrow \{\alpha = 0\}$ and $\{\alpha = \mathsf{F}(\beta,\beta)\} \to \{\alpha = 0\}$ are non-left-linear. For example, the left-hand side of a left-linear rule has to be a tree.

Before stating the definition of overlapping rules we need some more definitions.

Definition 7.3 A *substitution* σ is a map from Term(Σ) to Term(Σ) such that

$$\sigma(\mathsf{F}(t_1,\cdots,t_n)) = \mathsf{F}(\sigma(t_1),\cdots,\sigma(t_n)).$$

We extend σ to system of recursion equations as follows (we will require that σ only acts non trivially on the free variables of the system): $\sigma(\{\alpha_0 = t_0,\ \cdots,\alpha_n = t_n\}) = \{\sigma(\alpha_0) = \sigma(t_0),\ \cdots,\sigma(\alpha_n) = \sigma(t_n)\}$. We will also write t^σ instead of $\sigma(t)$.

Definition 7.4 Let g_1, g_2 be term graphs. g_1 and g_2 are called *compatible* (written as $g_1 \uparrow g_2$) if there exists a term graph g_3, substitutions σ_1 and σ_2, such that
(i) $g_1^{\sigma_1} \subseteq g_3$ and $g_2^{\sigma_2} \subseteq g_3$
(ii) $\mathtt{ROOT}(g_1^{\sigma_1}) = \mathtt{ROOT}(g_2^{\sigma_2}) = \mathtt{ROOT}(g_3)$.

Definition 7.5 Let $\tau_1 : l_1 \rightarrow r_1, \tau_2 : l_2 \rightarrow r_2$. We say that τ_1 *overlaps with* τ_2 iff $\exists\alpha$ occurring in l_1, such that

$$(l_1 \mid \alpha) \uparrow l_2$$

If $\tau_1 = \tau_2$, then it must be the case that α is distinct from the root of l_1.

Example 7.6 Rule $\tau : l \equiv \{\alpha = \mathsf{L}(\beta),\ \beta = \mathsf{L}(\delta)\} \longrightarrow \{\alpha = 0\}$ overlaps with itself because $(l \mid \beta) \uparrow l$. The rule $\{\alpha = \mathsf{L}(\mathsf{L}(\delta)\} \longrightarrow \{\alpha = 0\}$ is not overlapping. Likewise the rules: $\{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\delta)\} \longrightarrow \{\alpha = 0\}$ and $\{\alpha = \mathsf{H}(\beta),\ \beta = \mathsf{G}(\delta)\} \longrightarrow \{\alpha = 0\}$ are not overlapping.

In the following, TGRS stands for Term Graph Rewriting System.

Definition 7.7 A TGRS is said to be *orthogonal* if all the rules are left-linear and non-overlapping.

Definition 7.8 Let $\tau : l \rightarrow r$, α a bound variable occurring in g. Then (τ, α, σ) is a redex if $l^\sigma \subseteq g$ and $\mathtt{ROOT}(l^\sigma) = \alpha$.

Thus, detection of a redex boils down to matching parts of a system of recursion equations. If α is the root of a redex in g we also write $g \equiv C[\alpha = t]$, where $C[\Box]$ is the usual notation for a context.

Definition 7.9 Let (τ, α, σ) be a redex occurring in g. Let $\tau : l \rightarrow r$, $g \equiv C[\alpha = t]$. Then $g \longrightarrow h \equiv C[\alpha = (r^\sigma)']$, with $(r^\sigma)'$ denoting the renaming of all bound variables (using fresh variables), except the root, of r^σ.

Note that only the root equation gets rewritten, and that it may be replaced by several equations. Renaming is necessary to avoid collision with the variables in a system.

Example 7.10

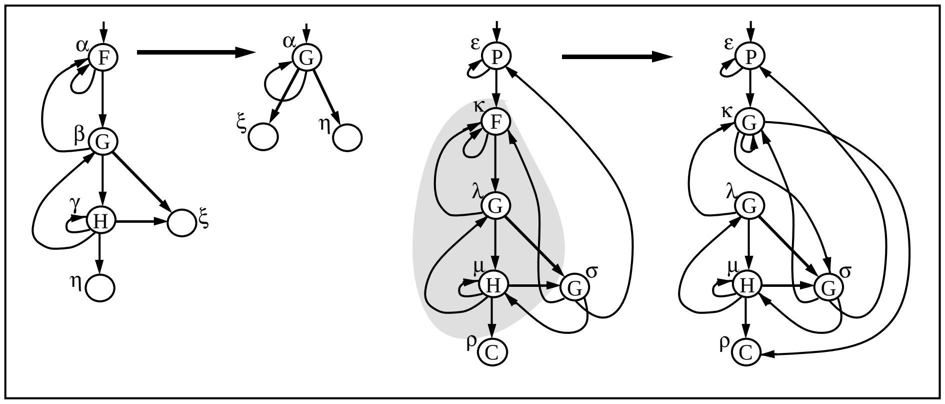


Figure 27.

(i) Consider the rule:

$$\{\alpha = \mathsf{F}(\alpha,\beta),\ \beta = \mathsf{G}(\alpha,\gamma,\xi),\ \gamma = \mathsf{H}(\gamma,\beta,\eta,\xi)\} \longrightarrow \{\alpha = \mathsf{G}(\xi,\alpha,\eta)\}$$

and the system:

$$\{\varepsilon = \mathsf{P}(\varepsilon,\kappa),\ \kappa = \mathsf{F}(\kappa,\lambda),\ \lambda = \mathsf{G}(\kappa,\mu,\sigma),\ \sigma = \mathsf{G}(\kappa,\varepsilon,\mu),\ \mu = \mathsf{H}(\mu,\lambda,\rho,\sigma),\ \rho = C\}$$

Both rule and system are displayed from left to right, respectively, in Figure 27. After 1-1 renaming the rule we obtain:

$$\{\kappa = \mathsf{F}(\kappa,\lambda),\ \lambda = \mathsf{G}(\kappa,\mu,\sigma),\ \mu = \mathsf{H}(\mu,\lambda,\rho,\sigma)\} \longrightarrow \{\kappa = \mathsf{G}(\sigma,\kappa,\rho)\}$$

We recognize the left-hand-side of the rule as a subset of the system under consideration (underlined part):

$$\{\varepsilon = \mathsf{P}(\varepsilon,\kappa),\ \underline{\kappa = \mathsf{F}(\kappa,\lambda)},\ \underline{\lambda = \mathsf{G}(\kappa,\mu,\sigma)},\ \sigma = \mathsf{G}(\kappa,\varepsilon,\mu),\ \underline{\mu = \mathsf{H}(\mu,\lambda,\rho,\sigma)},\ \rho = C\}$$

and rewrite accordingly, replacing only the first line (the root equation) $\underline{\kappa = \mathsf{F}(\kappa,\lambda)}$ of the redex by the right-hand side $\kappa = \mathsf{G}(\sigma,\kappa,\rho)$ of the rule (which in this example happens to be just one equation):

$$\{\varepsilon = \mathsf{P}(\varepsilon,\kappa),\ \kappa = \mathsf{G}(\sigma,\kappa,\rho),\ \lambda = \mathsf{G}(\kappa,\mu,\sigma),\ \sigma = \mathsf{G}(\kappa,\varepsilon,\mu),\ \mu = \mathsf{H}(\mu,\lambda,\rho,\sigma),\ \rho = C\}$$

In this case, no garbage collection, *i.e.,* , removal of superfluous equations, is necessary.

(ii) In the previous example, matching was done on the basis of 1-1 matching (renaming) of variables, but we want to be able to rewrite also *e.g.,* $\{\alpha = \mathsf{F}(\beta,\beta,\delta),\ \beta = \mathsf{G}(\beta)\}$ with the rewrite rule:

$$\{\xi = \mathsf{F}(\kappa,\lambda,\eta),\ \kappa = \mathsf{G}(\sigma),\ \lambda = \mathsf{G}(\tau)\} \longrightarrow \{\xi = \mathsf{G}(\lambda,\rho),\ \rho = \mathsf{H}(\eta,\xi,\tau)\}$$

This is possible, with the matching (variable substitution) $\xi \to \alpha, \kappa \to \beta, \lambda \to \beta, \eta \to \delta, \sigma \to \beta, \tau \to \beta$, which this time is not 1-1, we get the result

$$\{\alpha = \mathsf{G}(\beta,\rho),\ \rho = \mathsf{H}(\delta,\alpha,\beta),\ \beta = \mathsf{G}(\beta)\}$$

(iii) To allow for reduction of non-flat systems, a term (*i.e.,* TRS term) can be substituted for the free variables of a rule. Thus, the rule:

$$\{\alpha = \mathsf{F}(\beta)\} \to \{\alpha = \beta\}$$

is applicable to $g \equiv \{\psi = \mathsf{F}(\mathsf{G}(0))\}$, with substitution: $\alpha \to \psi,\ \beta \to \mathsf{G}(0)$. After reduction we will obtain: $\{\psi = \mathsf{G}(0)\}$.

(iv) Consider the following rules:

$$\tau_1 :\ \{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\delta)\} \longrightarrow \{\alpha = \mathsf{H}(\delta)\} \quad \text{and} \quad \tau_2 :\ \{\alpha = \mathsf{F}(\mathsf{G}(\delta))\} \longrightarrow \{\alpha = \delta\}$$

and the following systems:

$$g_1 \equiv \{\psi = \mathsf{F}(\eta),\ \eta = \mathsf{G}(\phi),\ \phi = 0\} \quad g_2 \equiv \{\psi = \mathsf{F}(\mathsf{G}(0))\} \quad g_3 \equiv \{\psi = \mathsf{F}(\mathsf{G}(\phi)),\ \phi = 0\}$$

Rule τ_1 is applicable to g_1 but not to g_2 and g_3 because it involves matching β with either $\mathsf{F}(\mathsf{G}(0))$ or $(\mathsf{G}(\phi))$. On the other hand, rule τ_2 is applicable to both g_2 and g_3, but not to g_1.

Theorem 7.11 *A TGRS without overlapping rules is confluent up to renaming.*
Proof: Let $(\alpha, \sigma_1, \tau_1)$ and $(\beta, \sigma_2, \tau_2)$ be two redexes occurring in g ($\tau_i : l_i \longrightarrow r_i (i = 1, 2)$). Let $g \xrightarrow{\alpha} g_1$ and $g \xrightarrow{\beta} g_2$. Since all rules are non overlapping it must be the case that $\alpha \not\equiv \beta$. Moreover, the descendant of α in g_2 is still a redex, likewise, for β in g_1. Therefore, the following diagram trivially commutes.

$$\begin{tikzcd}
g \equiv C[\alpha = t, \beta = s] \arrow[r] \arrow[d] & g_2 \equiv C[\alpha = t, \beta = ({r_2}^{\sigma_2})'] \arrow[d] \\
g_1 \equiv C[\alpha = ({r_1}^{\sigma_1})', \beta = s] \arrow[r] & g_3 \equiv C[\alpha = ({r_1}^{\sigma_1})', \beta = ({r_2}^{\sigma_2})']
\end{tikzcd}$$

□

The restriction of non-left-linearity is not necessary to guarantee confluence. However, we do need to restrict our attention to orthogonal TGRSs if also copying is considered, as observed in [Sme93] for the acyclic case.

Example 7.12 Consider the non-left-linear rule τ: $\{\alpha = \mathsf{F}(\gamma, \gamma)\} \longrightarrow \{\alpha = 1\}$ and the system $g \equiv \{\eta = \mathsf{F}(\beta, \beta),\ \beta = 1\}$. Rule τ is applicable to g. However, if we perform one copy step obtaining g_1:

$$g \longrightarrow_{\mathsf{C}} g_1 \equiv \{\eta = \mathsf{F}(\beta, \beta'),\ \beta = 1,\ \beta' = 1\}$$

then rule τ is no longer applicable to g_1.

The proof of the following proposition is routine.

Proposition 7.13 *Given an orthogonal TGRS, then:*

$$g \longrightarrow g_1 \text{ and } g \longrightarrow_{\mathsf{C}} g_2 \Longrightarrow \exists g_3, g_2 \twoheadrightarrow g_3 \text{ and } g_1 \longrightarrow_{\mathsf{C}} g_3.$$

Pictorially:

$$\begin{tikzcd}
g \arrow[r] \arrow[d, "\mathsf{C}"] & g_1 \arrow[d, dashed, "\mathsf{C}"] \\
g_2 \arrow[r, two heads] & g_3
\end{tikzcd}$$

(Here $\twoheadrightarrow$ is the transitive reflexive closure of $\longrightarrow$.)

Remark 7.14 Reduction $\twoheadrightarrow$ does not commute with $\underset{\mathsf{C}}{\longleftarrow}$ (or $\underline{\rightarrow}$, functional bisimulation). Counterexample: consider the rule

$$\{\alpha = \mathsf{C}\} \longrightarrow \{\alpha = \mathsf{D}\}$$

Then $g_0 \equiv \{\alpha = \mathsf{F}(\beta, \gamma),\ \beta = \mathsf{C},\ \gamma = \mathsf{C}\} \longrightarrow \{\alpha = \mathsf{F}(\beta, \gamma),\ \beta = \mathsf{C},\ \gamma = \mathsf{D}\} \equiv g_1$. Also $g_0 \underset{\mathsf{C}}{\longleftarrow} \{\alpha = \mathsf{F}(\beta, \beta),\ \beta = \mathsf{C}\} \equiv g_2$. Now g_2 can be rewritten to $g_3 \equiv \{\alpha = \mathsf{F}(\beta, \beta),\ \beta = \mathsf{D}\}$, but that is not a $\underset{\mathsf{C}}{\longleftarrow}$ -result of g_1. (See Figure 28). So, reduction $\longrightarrow$ does not commute with bisimilarity $\underline{\leftrightarrow}$. Yet, many interesting facts can be established for this union $\twoheadrightarrow \cup \underset{\mathsf{C}}{\longleftarrow}$; this has been established in work of Plump and Hofmann ("collapsed tree rewriting") [HP88, Plu93]. There, after each rewriting the graph can be maximally collapsed; this yields a gain in efficiency.

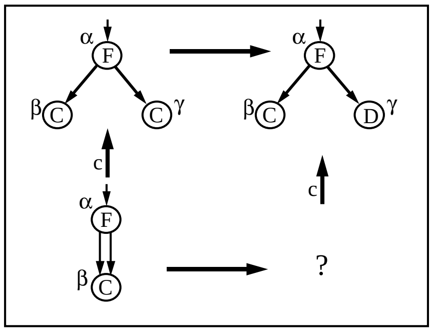


Figure 28.

Corollary 7.15 *Orthogonal TGRSs are confluent with respect to* $\longrightarrow \cup \longrightarrow_{\mathsf{c}}$.

Proof: At once from Theorem 7.11 and Proposition 7.13. □

Proposition 7.16 *Orthogonal TGRSs are confluent with respect to* $\longrightarrow \cup \longrightarrow_{\mathsf{csf}}$.

7.1 Weakly orthogonal term graph rewriting

It is well-known that for first-order term rewriting one can release the orthogonality condition somewhat while still retaining confluence. Specifically, all weakly orthogonal TRS are confluent. A TRS is called *weakly orthogonal* if all critical pairs are trivial, *i.e.,* of the form $\langle t, t\rangle$. This is a useful result, since it admits also rewrite rules such as:

$$\begin{array}{lcl} \mathsf{or}(x, \mathsf{true}) & \longrightarrow & \mathsf{true} \\ \mathsf{or}(\mathsf{true}, x) & \longrightarrow & \mathsf{true} \end{array}$$

(Recently, this has also been shown to be the case for higher-order term rewriting in the framework of Combinatory Reduction Systems or CRSs: all weakly orthogonal CRSs are confluent. The prime example of a weakly orthogonal CRS is $\lambda\beta\eta$-calculus. This result is due to van Oostrom and van Raamsdonk.)

The question arises whether analogously we can upgrade the confluence theorem for orthogonal TGRSs to allow for rules that are only weakly orthogonal. To that end, we note that the notion does not carry over directly to TGRSs.

Example 7.17 Consider the TGRS system R with rules:

$$\begin{array}{lcl} \tau_1 : \{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\eta)\} & \longrightarrow & \{\alpha = \mathsf{F}(\psi),\ \psi = 0\} \\ \tau_2 : \{\alpha = \mathsf{G}(\beta)\} & \longrightarrow & \{\alpha = 0\} \end{array}$$

If we apply naively the familiar notion of weakly orthogonality we erroneously deduce that the above rules are weakly orthogonal and thus deriving that R is confluent. However, consider the term:

$$M \equiv \{\alpha = \mathsf{H}(\beta, \eta),\ \beta = \mathsf{F}(\eta),\ \eta = \mathsf{G}(\varepsilon),\ \varepsilon = 0\}$$

Then we have the following reductions:

$$\begin{aligned} M \xrightarrow[\tau_1]{} \ & M_1 \equiv \{\alpha = \mathsf{H}(\beta,\eta),\ \beta = \mathsf{F}(\psi), \psi = 0,\ \eta = \mathsf{G}(\varepsilon),\ \varepsilon = 0\} \xrightarrow[\tau_1]{} \\ & M_1' \equiv \{\alpha = \mathsf{H}(\beta,\eta),\ \beta = \mathsf{F}(\psi), \psi = 0,\ \eta = 0\} \\ M \xrightarrow[\tau_1]{} & M_2 \equiv \{\alpha = \mathsf{H}(\beta,\eta),\ \beta = \mathsf{F}(\eta), \eta = 0\} \end{aligned}$$

Note that M_1 and M_2 are not equivalent up to renaming; note however that $M_2 \longrightarrow_{\mathsf{c}} M_1'$.

From this example we conclude that in analyzing the critical pairs we cannot discard the garbage in a term. That is, let M be the common instance of rules τ_1 and τ_2, *i.e.,* $M \equiv (\tau_1 \mid \alpha) \uparrow \tau_2$, with $M \equiv \{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\gamma),\ \gamma = \delta\}$. We consider the critical pair $\langle\{\alpha = \mathsf{F}(\psi),\ \psi = 0,\ \beta = \mathsf{G}(\gamma),\ \gamma = 0\}, \{\alpha = \mathsf{F}(\beta),\ \ \beta = 0,\ \gamma = \delta\}\rangle$, not trivial because the two terms are not equivalent up to renaming.

Interestingly enough the rules:

$$\begin{aligned} \{\alpha = \mathsf{F}(\mathsf{G}(\beta))\} & \longrightarrow \{\alpha = \mathsf{F}(0)\} \\ \{\alpha = \mathsf{G}(\delta)\} & \longrightarrow \{\alpha = 0\} \end{aligned}$$

are not even overlapping.

Definition 7.18 Given two rules τ_1 and τ_2, we say that τ_1 *overlaps weakly with* $\tau_2 \Longleftrightarrow$ if τ_1 overlaps with τ_2 then if we let M be the common instance of τ_1 and τ_2, we have $M \xrightarrow[\tau_1]{} M_1$ and $M \xrightarrow[\tau_2]{} M_2$, with M_1 and M_2 equivalent up to renaming without removing the garbage.

Proposition 7.19 *A TGRS with weakly overlapping rules is confluent.*

Note that if we apply the naive version of weakly orthogonality then the system is confluent up to copying.

Definition 7.20 A TGRS is said to be *weakly orthogonal* iff all the rules are left-linear and weakly overlapping.

8. Translation of a TRS into a TGRS

8.1 Flat translation

In this section we will not study the relation between 'ordinary' term rewriting and term graph rewriting; we only briefly indicate the straightforward way in which TRSs can be translated into TGRSs, thereby preserving orthogonality. This translation yields an acyclic TGRS (*i.e.,* left-hand side, right-hand side of all rules are acyclic). For a comparison between TRSs and TGRSs with respect to obtaining normal forms, in the acyclic setting, see [BvEG$^+$87]. For general notions of adequacy of graph rewriting versus term rewriting, see [KKSdV94, Ari93].

Definition 8.1 Let R be a TRS, $\tau : l(\vec{x}) \longrightarrow r(\vec{x})$ a rule in R. Then τ will be translated as follows:

(i) replace $\vec{x} = x_1, \cdots, x_n$ by recursion variables $\vec{\alpha} = \alpha_1, \cdots, \alpha_n$;

(ii) consider the so obtained TRS rule

$$\{\alpha_0 = l(\vec{\alpha})\} \longrightarrow \{\alpha_0 = r(\vec{\alpha})\}$$

Now the intended graph rewriting rule is obtained by flattening left-and-right hand side of the last rule:

$$\text{flat}(\{\alpha_0 = l(\vec{\alpha})\}) \longrightarrow \text{flat}(\{\alpha_0 = r(\vec{\alpha})\}).$$

It should be pointed out that with the resulting TGRS, we are also able to rewrite cyclic graphs. That is, we not only profit from the 'horizontal' sharing of subterms as in acyclic graphs, but also from being able to rewrite cyclic terms. Now we have a link with transfinite orthogonal term rewriting, as developed in [KKSdV95]. Without the routine proofs we mention the following 'soundness' property of graph rewriting with respect to infinitary rewriting.

Let R be an orthogonal TRS. Let R_γ be its graph version as defined above. Let R^∞ be the infinitary version of R. (Actually, the rules of R^∞ are as for R, but the instantiations may involve infinite terms.)

Let $[\![\ ]\!]$ be the possibly infinite unwinding of a graph in R_γ, yielding a tree in R^∞.
Then:
(i) $g \longrightarrow_{\mathsf{csf}} g' \Longrightarrow [\![g]\!] = [\![g']\!]$
(ii) $g \longrightarrow_{R_\gamma} g' \Longrightarrow [\![g]\!] \longrightarrow_{\leq\omega} [\![g']\!]$.
Here, in (ii), the left-hand side is a graph rewriting step, while the right-hand side is a possibly infinite rewriting sequence in R^∞. Soundness with respect to infinitary rewriting is also proved in [FW91].

Note that every orthogonal acyclic flat TGRS, is the image under this translation of an (orthogonal) TRS.

Example 8.2 (i) Consider the rules of Combinatory Logic:

$$\begin{array}{lcl} \mathsf{Ap}(\mathsf{Ap}(\mathsf{Ap}(\mathsf{S},x),y,z)) & \longrightarrow & \mathsf{Ap}(\mathsf{Ap}(x,z),\mathsf{Ap}(y,z)) \\ \mathsf{Ap}(\mathsf{Ap}(\mathsf{K},x),y) & \longrightarrow & x \end{array}$$

Pictorially the S-rule is drawn in Figure 29; in Figure 30 we show which nodes are shared.

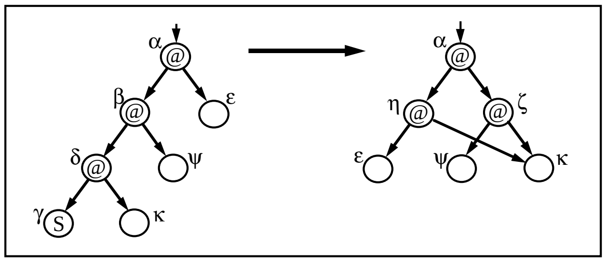


Figure 29.

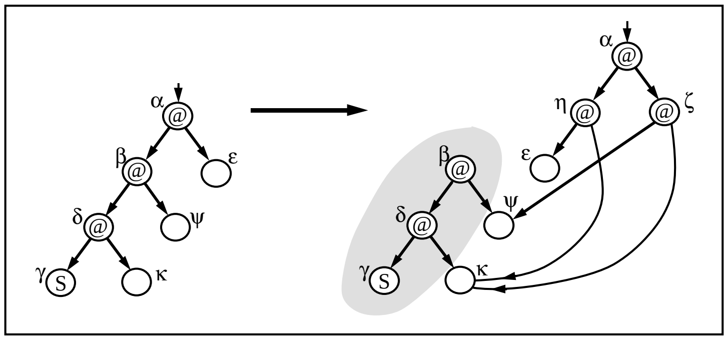


Figure 30.

These will be translated into a TGRS in the following way:

$$\begin{array}{lcl} \{\alpha = \mathsf{Ap}(\beta,\varepsilon),\ \beta = \mathsf{Ap}(\delta,\psi),\ \delta = \mathsf{Ap}(\phi,\kappa),\ \phi = \mathsf{S}\} & \longrightarrow & \{\alpha = \mathsf{Ap}(\gamma,\eta),\ \gamma = \mathsf{Ap}(\kappa,\varepsilon),\ \eta = \mathsf{Ap}(\psi,\varepsilon)\} \\ \{\alpha = \mathsf{Ap}(\beta,\gamma),\ \beta = \mathsf{Ap}(\phi,\delta),\ \phi = \mathsf{K}\} & \longrightarrow & \{\alpha = \gamma\} \end{array}$$

(ii) Consider Combinatory Logic with the rule $\mathsf{D}xx \longrightarrow x$. This TRS is not confluent [Klo], but its TGRS translation is. The translation consists of the TGRS in the preceding examples, plus

$$\{\alpha = \mathsf{Ap}(\beta,\gamma),\ \beta = \mathsf{Ap}(\delta,\gamma),\ \delta = \mathsf{D}\} \longrightarrow \{\alpha = \gamma\}.$$

Confluence follows from Theorem 7.11; the rules do not overlap.

Example 8.3 Consider the TRS rules:

$$\begin{array}{lcl} \mathsf{C} & \longrightarrow & \mathsf{A}(\mathsf{B}(\mathsf{C})) \\ \mathsf{A}(x) & \longrightarrow & x \\ \mathsf{B}(x) & \longrightarrow & x \end{array}$$

then by applying the translation we obtain:

$$\begin{array}{lcl} \{\alpha = \mathsf{C}\} & \longrightarrow & \{\alpha = \mathsf{A}(\beta),\ \beta = \mathsf{B}(\delta),\ \delta = \mathsf{C}\} \\ \{\alpha = \mathsf{A}(\beta)\} & \longrightarrow & \{\alpha = \beta\} \\ \{\alpha = \mathsf{B}(\beta)\} & \longrightarrow & \{\alpha = \beta\} \end{array}$$

Remark 8.4 Note that we could apply a more optimized translation, thus avoiding the creation of a new redex each time the first rule is applied. That is, we can translate the first rule as:

$$\{\alpha = \mathsf{C}\} \longrightarrow \{\alpha = \mathsf{A}(\beta),\ \beta = \mathsf{B}(\alpha)\}$$

This translation exhibits the "redex capturing " phenomenon discussed in Farmer and Watro [FW91].

We will consider this optimization not as part of the basic graph rewriting mechanism, but rather as an 'add-on feature" whose effect will not be studied in the present paper.

Proposition 8.5 *The translation of an orthogonal TRS is an orthogonal TGRS.*

Proof: Clear. □

As mentioned in the previous section the translation of a weakly orthogonal TRS is not necessarily a weakly orthogonal TGRS.

Example 8.6 Let R be a TRS with rules:

$$\begin{array}{lcl} \mathsf{F}(\mathsf{G}(x)) & \longrightarrow & \mathsf{F}(0) \\ \mathsf{G}(x) & \longrightarrow & 0 \end{array}$$

R is weakly orthogonal because the critical pair $\langle \mathsf{F}(\mathsf{G}(x)), \mathsf{G}(x) \rangle$ has a common reduct, namely the term $\mathsf{F}(0)$. While the translated system R':

$$\begin{array}{lcl} \{\alpha = \mathsf{F}(\beta),\ \beta = \mathsf{G}(\eta),\ \eta = x\} & \longrightarrow & \{\alpha = \mathsf{F}(\psi),\ \psi = 0\} \\ \{\alpha = \mathsf{G}(\beta),\ \beta = x\} & \longrightarrow & \{\alpha = 0\} \end{array}$$

is not weakly orthogonal (as already pointed out in the previous section).

8.2 Non-flat translation

Proposition 8.7 *The non-flat translation of a weakly orthogonal TRS is a weakly orthogonal TGRS.*

9. Concluding remarks and future work

Some of the simplicity of term rewriting is lost in dealing with term graphs due to the matching of sets of equations instead of matching of terms. Furthermore, the natural operation of substitution introduces non-confluence. This loss of confluence is the more surprising in a comparison with the confluent R_μ-calculus, where also a form of substitution is present in order to create redexes. This raises the desire of finding a calculus for term graph rewriting that combines the simplicity of term rewriting with the ability to express the different forms of sharing that arise in common implementations of functional languages (*i.e.,* horizontal and vertical sharing). Presently we are elaborating a framework employing nested systems of recursion equations that seems promising in this respect. To design and understand such a framework, an analysis of fundamental operations on term graphs such as copying, substitution, flattening and hiding as in Section 4 of this paper is indispensable.

We are also interested in extending the framework to accommodate cyclic λ-graphs, an endeavor that can be seen as an extension of the work on the $\lambda\sigma$-calculi (λ-calculi with explicit substitution) [ACCL91, Cur93, HL89, Les94, Ros92]. However, the latter concern acyclic substitutions only, while we aim at a calculus allowing cyclic substitutions. A preliminary study appears in [AK94].

Furthermore, we intend to study the suitability of equational graph rewriting for expressing side-effect operations. To that end, an extension is needed to include both rules and terms with multiple roots.

We expect that a final system obtained along these lines, can be used to express the operational semantics of current functional languages extended with a notion of state [HPJW$^+$92, Nik91].

Acknowledgements
This work was done at the Department of Software Technology of CWI, at the Department of Computer Science of the Free University, and at the Department of Computer and Information Science of the University of Oregon. Zena Ariola thanks both CWI and the Free University to make her visits possible. Jan Willem Klop thanks the University of Oregon for funding his stay in April '93.

We thank Ronan Sleep, Richard Kennaway, Fer-Jan de Vries, Paola Inverardi, Andrea Corradini, Stefan Blom for stimulating discussions about this work, and Robin Milner for pointing out an important reference concerning fixed-point equations.

References

[AA93] Z. M. Ariola and Arvind. Graph rewriting systems for efficient compilation. In M R. Sleep, M J. Plasmeijer, and M C D J. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, pages 77–90. John Wiley & Sons, 1993.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[AK94] Z. M. Ariola and J. W. Klop. Cyclic lambda graph rewriting. In *Proc. Ninth Symposium on Logic in Computer Science (LICS'94), Paris, France*, pages 416–425, 1994.

[Ari92] Z. M. Ariola. *An Algebraic Approach to the Compilation and Operational Semantics of Functional Languages with I-structures.* PhD thesis, MIT Technical Report TR-544, 1992.

[Ari93] Z. M. Ariola. Relating graph and term rewriting via Böhm models. In C Kirchner, editor, *Proc. 5th International Conference on Rewriting Techniques and Applications (RTA-93), Montreal, Canada, Springer-Verlag LNCS 690*, pages 183–197, 1993.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[BE91] R. V. Book (Editor). *Proc. 4th International Conference on Rewriting Techniques and Applications (RTA-91), Como, Italy, Springer-Verlag LNCS 488.* 1991.

[BS93] E. Barendsen and J. E. W. Smetsers. Conventional and uniqueness typing in graph rewrite systems. Technical Report CSI-R9328, Computing Science Institute, University of Nijmegen, 1993.

[BvEG$^+$87] H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *Proc. Conference on Parallel Architecture and Languages Europe (PARLE '87), Eindhoven, The Netherlands, Springer-Verlag LNCS 259*, pages 141–158, 1987.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra.* Cambridge University Press, 1990.

[CKV74] B. Courcelle, G. Kahn, and J. Vuillemin. Algorithmes d'équivalence et de réduction à des expressions minimales dans une classe d'équations récursives simples. In J. Loeckx, editor, *Proc. 2nd Colloquium on Automata, Languages and*

Programming (ICALP '74), University of Saarbrucken, Springer-Verlag LNCS 14, pages 200–213, 1974.

[Cor93] A. Corradini. Term rewriting in CT_{Σ}. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. Colloquium on Trees in Algebra and Programming (CAAP '93), Springer-Verlag LNCS 668*, pages 468–484, 1993.

[Cou78] B. Courcelle. On recursive equations having a unique solution. Technical Report 285, IRIA Rapport de Recherche, 1978.

[Cou90] B. Courcelle. Recursive applicative program schemes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 461–492. Elsevier - The MIT Press, 1990.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential algorithms, and Functional Programming.* Birkhäuser, 2nd edition, 1993.

[CV76] B. Courcelle and J. Vuillemin. Completeness results for the equivalence of recursive schemas. *JCSS*, 12:179–197, 1976.

[dB] J. W. de Bakker. *Recursive procedures.* Mathematical Centre Tracts 24, Mathematisch Centrum, Amsterdam, 1971.

[dB80] J. W. de Bakker. *Mathematical Theory of Program Correctness.* Prentice Hall International. Series in Computer Science, 1980.

[DJ90] N. Dershowitz and J. P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier - The MIT Press, 1990.

[Far90] W. M. Farmer. A correctness proof for combinator reduction with cycles. *ACM Transactions on Programming Languages and Systems*, 12(1):123–134, 1990.

[FW91] W. M. Farmer and R. J. Watro. Redex capturing in term graph rewriting. In R. V. Book, editor, *Proc. 4th International Conference on Rewriting Techniques and Applications (RTA-91), Como, Italy, Springer-Verlag LNCS 488*, pages 13–24, 1991.

[GAL92] G. Gonthier, M. Abadi, and J.-J Lévy. The geometry of optimal lambda reduction. In *Proc. ACM Conference on Principles of Programming Languages*, 1992.

[GKS90] J. R. W. Glauert, J. R. Kennaway, and M. R. Sleep. Dactl: An experimental graph rewriting language. In *Proc. 4th International Workshop on Graph Grammars and their Application to Computer Science, Bremen, Germany, Springer-Verlag LNCS 532*, pages 378–395, 1990.

[Gue81] I. Guessarian. *Algebraic Semantics, Springer-Verlag LNCS 99.* 1981.

[HL89] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium, Izu*, 1989.

[HP88] B. Hoffman and D. Plump. Jungle evaluation for efficient term rewriting. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proc. International Workshop on Algebraic and Logic Programming, Springer-Verlag LNCS 343*, pages

191–203, 1988.

[HPJW$^+$92] P. Hudak, S. Peyton Jones, P. Wadler, B. Boutel, J. Fairbairn, J. Fasel, K. Hammond, J. Hughes, T. Johnsson, D. Kieburtz, R. Nikhil, W. Partain, and J. Peterson. Report on the programming language Haskell. *ACM SIGPLAN Notices*, 27(5):1–64, 1992.

[Kat90] V. K. Kathail. *Optimal Interpreters for Lambda-calculus Based Functional Languages.* PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1990.

[Ken87] J. R. Kennaway. On graph rewriting. *Theoretical Computer Science*, 52:37–58, 1987.

[Ken88] J. R. Kennaway. Corrigendum on 'on graph rewriting'. *Theoretical Computer Science*, 61:317–320, 1988.

[Ken90] J. R. Kennaway. Graph rewriting on some categories of partial morphisms. In *Proc. 4th International Workshop on Graph Grammars and their Application to Computer Science, Bremen, Germany, Springer-Verlag LNCS 532*, pages 490–504, 1990.

[KKSdV94] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. The adequacy of term graph rewriting for simulating term rewriting. *Transactions on Programming Languages and Systems*, 16(3):493–523, 1994.

[KKSdV95] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *To appear in Information and Computation.* 1995.

[Klo] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Mathematical Centre Tracts 127, CWI, Amsterdam,1980.

[Klo92] J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume II, pages 1–116. Oxford University Press, 1992.

[KvOvR93] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, 1993. A Collection of Contributions in Honour of Corrado Böhm on the Occasion of his 70th Birthday, guest eds. M. Dezani-Ciancaglini, S. Ronchi Della Rocca and M. Venturini-Zilli.

[Lé80] J.-J. Lévy. Optimal reductions in the lambda-calculus. In *To H-.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 159–291. Academic Press, 1980.

[Lö93] M. Löwe. Algebraic approach to single pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.

[Lam90] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. ACM Conference on Principles of Programming Languages, San Francisco*, 1990.

[Les94] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. In *Proc. 21st Symposium on Principles of Programming Languages (POPL '94),* *Portland, Oregon*, pages 60–69, 1994.

[LV93] N. A. Lynch and F. W. Vaandrager. Forward and backward simulations, part i: untimed systems. Technical Report CS-R9313, CWI, Amsterdam, 1993.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *JCSS*, 28:227–247, 1984.

[Mil89] R. Milner. *Communication and concurrency.* Prentice Hall International, 1989.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[Nik91] R. S. Nikhil. Id (version 90.1) reference manual. Technical Report 284-2, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, 1991.

[Plu93] D. Plump. *Evaluation of Functional Expressions by Hypergraph Rewriting.* PhD thesis, Universität Bremen, 1993.

[PvE93] M. J. Plasmeijer and M. C. J. D. van Eekelen. *Functional Programming and Parallel Graph Rewriting.* Addison Wesley, 1993.

[Rao84] J. C. Raoult. On graph rewritings. *Theoretical Computer Science*, 32:1–24, 1984.

[Ros92] K. H. Rose. Explicit cyclic substitutions. In M. Rusinowitch and J. L. Rémy, editors, *Proc. 3rd International Workshop on Conditional Term Rewriting Systems (CTRS-92), Pont-à-Mousson, France, Springer-Verlag LNCS 656*, pages 36–50, 1992.

[Sme93] J. E. W. Smetsers. *Graph Rewriting and Functional Languages.* PhD thesis, University of Nijmegen, 1993.

[SPvE93] M. R. Sleep, M. J. Plasmeijer, and M. C. D. J. van Eekelen, editors. *Term Graph Rewriting: Theory and Practice.* John Wiley & Sons, 1993.

[SS93] T. Schmidt and T. Ströhlein. *Relations and Graphs.* EATCS Monographs on Theoretical Computer Science Springer-Verlag, 1993.

[Tur79] D. A. Turner. A new implementation technique for applicative languages. *Software Practice and Experience*, 9:31–49, 1979.

[vR93] F. van Raamsdonk. Confluence and superdevelopments. In C Kirchner, editor, *Proc. 5th International Conference on Rewriting Techniques and Applications (RTA-93), Montreal, Canada, Springer-Verlag LNCS 690*, pages 168–182, 1993.

[Vui73] J. E. Vuillemin. *Proof Techniques for Recursive Programs.* PhD thesis, Stanford University, 1973.

[Vui74] J. E. Vuillemin. *Syntaxe, Sémantique et Axiomatique d'un Langage de Programmation Simple.* PhD thesis, Université PARIS VI, 1974.

[Wad71] C. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* 1971. PhD thesis, University of Oxford.