Rational Unification in 28 Characters

Pietro Cenciarelli, and Alessandro Tiberi

Dipartimento di Informatica, University of Rome - "La Sapienza", Italy

Abstract

We present a case study where *Synchronising Graphs*, a system of parallel graph transformation, are used to solve the syntactic unification problem for first order rational terms (with possibly infinite unifier). The solution we offer is *efficient*, that is quasi-linear, and *simple*: a program of 28 characters.

Key words: synchronising graphs, synchronized hyperedge replacement, graph rewriting, unification, rational equations, regular trees, coalgebra.

1 Introduction

Graph-like structures have been used extensively in the literature by algorithms for *syntactic unification* [PW76,Cou83]. This is mainly because allowing terms to share common subterms often improves efficiency. Surprisingly, much less common is to see *graph rewriting* systems being used to solve unification. The problem of making two terms equivalent with respect to a congruence induced by equational axioms, called *equational* unification, has indeed been approached by using *term graph rewriting* [Plu99]. However, this is rather different from *syntactic* unification [Rob65], the equational case being in general undecidable, and further complicated by the possible absence of a *most general* unifier. What we present here is an efficient syntactic unification algorithm axiomatised in 28 characters within the system of *Synchronising Graphs*. And we prove its correctness.

Synchronising Graphs (SG) were proposed in [CTT04] as a model of process interaction in a network environment. The model is based on *hyperedge replacement* [DM87,FMT01], a form of graph transformation where edges, representing processes, interact by synchronising action and co-action pairs at specific synchronisation points, the nodes, representing communication channels. The behaviour of parallel, possibly distributed systems is specified in SG by a set of axioms. System transitions are derived by means of inference rules implementing agent interaction (synchronisation) and resource encapsulation (restriction). In [CT04,CTT04] suitable axiomatisations of synchronising graphs were proven operationally equivalent to the calculus of *Mobile*

©2005 Published by Elsevier Science B. V.

Ambients [CG00], to the *distributed CCS* [RH01], and to the *Fusion Calculus* [PV98]. These results support the view of Synchronising Graphs as a common semantic framework for interpreting different calculi for mobility. In the present paper we put SG at work on a different kind of application where "topological" aspects matter: in programming unification, terms play the role of processes, while the variables represent the shared network infrastructure.

The word "rational" in the title refers to *rational expressions* [Cou80], that is expressions denoting *regular* trees. These are the possibly infinite trees having a *finite* number of subtrees, and they are typically obtained by unraveling of possibly *cyclic* finite graphs. Although the starting point of our algorithm will be an equation $t = t'$, with *finite* t and t', we do offer a solution to equations such as $x = f(x)$, viz. $x = f(f(f(\dots)))$, which give rise to cyclic graphs. Hence, there is no loss of generality in assuming t and t' as being finite (this allows a simple *inductive* translation into synchronising graphs), and the approach could be easily extended, as will become evident, to arbitrary rational terms.

The unification problem was first extended to infinite terms in [Hue76]. Besides the gain in generality, there are pragmatical reasons for allowing programs which unify a variable to a term in which that variable occurs, something which the usual mathematical theory behind Logic Programming forbids. For example, the *SICStus Prolog* programming language [Pro95] avoids performing costly occurs-checks each time a variable is unified with a term, and allows manipulation of cyclic terms without looping. While most famous unification algorithms [PW76,MM82] fail when applied to cyclic terms, efficient (quasi-linear) unification algorithms exist which deal with such terms [Cou83,Jaf84]. It is with these that we match ours.

Synopsis. Synchronising graphs (SG) are presented in Section 2, and compared with the related system of synchronised hyperedge replacement [FMT01] in Section 3. In Section 4 we write a program (two axioms) of 28 characters in SG to solve the problem of syntactic unification, and show it at work with an example. The correctness of the proposed solution is proven in Section 5.

Notation. We often write function application without parentheses, that is fx instead of $f(x)$. We write $\boldsymbol{x}$ for a finite sequence $x_1, x_2, \dots x_n$. If $f \subseteq A \times B$ is a relation and $a \in A$, we write fa for the set $\{b \in B \mid (a, b) \in f\}$. The *domain* of f is the set $dom(f) = \{x \in A \mid \exists\, b \in B\,.\,(a, b) \in f\}$. If φ is an equivalence relation, we write $[x]_\varphi$ the equivalence class of an element x; or just $[x]$, when φ is understood.

2 Synchronising graphs

Most of the material in this section comes from [CT04], to which we refer for more detail and examples.

Graphs. Let $\mathcal{N}$ be a set of *nodes*, which we consider fixed throughout. A *graph* $G = (E, G, R)$ consists of a set E of *hyperedges* (or just *edges*), an attachment function $G : E \to \mathcal{N}^*$ and a set $R \subseteq |G|$ of nodes, called *restricted*, where

$$|G| = \{x \in \mathcal{N} \mid \exists\, e \in E \text{ s.t. } G\,e = x_1 \dots x_n \text{ and } x = x_i\}$$

is the set of nodes of the graph. When $G\,e = x_1 x_2 \dots x_n$ we call n the *arity* of e and say that the i-th *tentacle* of e is attached to x_i. We denote by $res\,(G)$ the set of restricted nodes of G, and by $fn\,(G)$ the set $|G| - res\,(G)$ of *free* nodes of G. We write $e(\boldsymbol{x})$ for an edge of a graph G such that $G\,e = \boldsymbol{x}$. Moreover, we let $\nu x\,G$ denote the graph $(E, G, R \cup \{x\})$ when $x \in |G|$, while $\nu x\,G = G$ otherwise. If (E, G, R) and (D, F, S) are graphs such that $E \cap D = |G| \cap S = |F| \cap R = \emptyset$, we write $G|F$ the graph $(E \cup D, G + F, R \cup S)$, whose attachment function $G + F$ maps $e \in E$ to $G\,e$ and $d \in D$ to $F\,d$.

Transitions. Let $Act = \{a, b, \dots\} \cup \{\overline{a}, \overline{b}, \dots\}$ be a set of *actions* and *co-actions* (overlined), and let $\overline{\overline{a}}$ denote a. We write Act^+ the set $Act \times \mathcal{N}^*$. Given $(a, \boldsymbol{x})$ in Act^+, we call the components of $\boldsymbol{x}$ *arguments* of a. A *pre-transition* Λ of a graph G to a graph H, written:

$$G \xrightarrow{\Lambda} H,$$

is a relation $\Lambda \subseteq \mathcal{N} \times Act^+$ such that $dom\,(\Lambda) \subseteq |G|$. We write $(x, a, \boldsymbol{y})$ for an element $(x, (a, \boldsymbol{y}))$ of Λ, and (x, a) when $\boldsymbol{y}$ is the empty sequence. Intuitively, $(a, \boldsymbol{y}) \in \Lambda\,x$ expresses the occurrence of action a at node x. In SG the occurrence of both $(a, \boldsymbol{y})$ and $(\overline{a}, \boldsymbol{z})$ at x triggers a synchronisation between two agents (edges) of the graph, what is traditionally represented by a *silent* action τ. When such is the case the synchronising agents may exchange information. This is implemented in SG by unifying the lists $\boldsymbol{y}$ and $\boldsymbol{z}$ of parameters, which are required to be of the same length. Only two agents at a time may synchronise at one node. Moreover, if an action occurs at a restricted node, then it *must* synchronise with a corresponding co-action, as we consider *observable* the unsynchronised actions. A restricted node may be "opened" by unifying it with an argument of an observable action, or with a node which is not restricted.

The above requirements are formalised as follows. An *action set* is a relation $\Lambda \subseteq \mathcal{N} \times Act^+$ such that, for all nodes x, $\Lambda\,x$ has *at most* two elements and, when so, it is of the form $\{(a, \boldsymbol{y}), (\overline{a}, \boldsymbol{z})\}$, where the lengths of vectors $\boldsymbol{y}$ and $\boldsymbol{z}$ coincide. Given an action set Λ, we denote by $\overset{\Lambda}{=}$ the smallest equivalence relation on nodes such that, if $(x, a, y_1 y_2 \dots y_n)$ and $(x, \overline{a}, z_1 z_2 \dots z_n)$ are in Λ, then $y_i \overset{\Lambda}{=} z_i$, for $i = 1 \dots n$. The *dangling* nodes of an action set Λ are arguments of unsynchronised actions. More precisely they are elements of the set $\{x \mid \exists\, y \text{ s.t. } \Lambda\,y = \{(a, z_1 \dots z_n)\} \text{ and } x \overset{\Lambda}{=} z_i\}$. A predicate $opens\,(\Lambda, x, G)$ is defined to hold precisely when either x is dangling in Λ or $[x]_{\overset{\Lambda}{=}} \not\subseteq res\,(G)$.

A *transition* is a pre-transition $G \xrightarrow{\Lambda} H$ such that:

(i) Λ is an action set;

(ii) if a node x is restricted in G then $|\Lambda\, x| \neq 1$;

(iii) if a node x occurs in H, then $x \in fn(H)$ if and only if $opens(\Lambda, x, G)$.

An *identity* is a transition of the form $G \xrightarrow{\emptyset} G$. We say that an action a is *observed* at node x during a transition Λ if $\Lambda\, x$ is a singleton $\{(a, \boldsymbol{y})\}$. The first clause above expresses the coherence of synchronisation; the second says that no actions can be observed at restricted nodes; the third states the conditions under which a node x, possibly restricted in G, can occur free in H.

Inference rules. Let $f : \mathcal{N} \to \mathcal{N}$ be a function on nodes and let (E, G, R) be a graph. We write fG the graph (E, fG, fR) obtained by substituting all nodes x in G with fx, that is, for all $e \in E$, if $G\, e = x_1 \dots x_n$ then $(fG)\, e = fx_1 \dots fx_n$. A function $f : \mathcal{N} \to \mathcal{N}$ is said to *agree* with an equivalence relation φ on $\mathcal{N}$ if, as a set of pairs, it is a subset of φ, that is if $(x, fx) \in \varphi$, for all nodes $x \in \mathcal{N}$. A *unifier* of φ is a function ρ which agrees with φ and such that $|\rho\,[x]| = 1$ for all x. By a slight abuse, we say that a function agrees with (or unifies) Λ to mean that it agrees with (unifies) $\stackrel{\Lambda}{=}$.

In SG, synchronisation is subject to a non-interference condition: two transitions $G \xrightarrow{\Lambda} H$ and $F \xrightarrow{\Theta} K$ can be synchronised provided they are disjoint and they share nodes only through their left sides. Formally, let $|\Lambda|$ denote the set $|G| \cup |H| \cup \{y \in \mathcal{N} \mid \exists\, x\,.\,(a, y_1 \dots y_n) \in \Lambda\, x \text{ and } y = y_i\}$; two transitions $G \xrightarrow{\Lambda} H$ and $F \xrightarrow{\Theta} K$ are said to be *non-interfering*, written $\Lambda \# \Theta$, when:

- $\Lambda \cap \Theta = \emptyset$, and moreover
- $|\Lambda| \cap |\Theta| = |G| \cap |F|$.

The rules of the system of synchronising graphs are:

$$[\,\text{sync}\,] \quad \frac{G \xrightarrow{\Lambda} H \qquad F \xrightarrow{\Theta} K}{G \mid F \xrightarrow{\Lambda \cup \Theta} \rho(H|K)} \quad (\text{if } \Lambda \# \Theta \text{ and } \rho \text{ unifies } \Lambda \cup \Theta)$$

$$[\,\text{open}\,] \quad \frac{G \xrightarrow{\Lambda} H}{\nu x\, G \xrightarrow{\Lambda} H} \quad (\text{if } opens(\Lambda, x, \nu x\, G))$$

$$[\,\text{res}\,] \quad \frac{G \xrightarrow{\Lambda} H}{\nu x\, G \xrightarrow{\Lambda} \nu\rho(x)\, \rho(H)} \quad (\text{if } \neg opens(\Lambda, x, \nu x\, G) \text{ and } \rho \text{ unifies } \Lambda)$$

Note that the applicability of [sync] is implicitly constrained by $\Lambda \cup \Theta$ being

a transition (e.g. Λ and Θ cannot issue different actions at a same node). Similar considerations apply to [open] and [res].

An *axiom* is a transition $G \xrightarrow{\Lambda} H$ such that $H = \rho H$ for some unifier ρ of Λ. This condition, stating that all nodes unified by Λ are fused in H, is preserved by the inference rules, and it is therefore satisfied by all transitions derived from axioms. Given a set $\mathcal{T}$ of axioms, a $\mathcal{T}$*-computation*, or just *computation* for short, is a sequence of transitions $G_0 \xrightarrow{\Lambda_1} G_1 \xrightarrow{\Lambda_2} \ldots$ each of which is derived from the axioms in $\mathcal{T}$.

Example. We conclude this section with an example where [sync] is used to move processes about. Since node restriction does not play a central role in the present paper, we refer to [CT04] for more intuition on [open] and [res].

A boolean *variable* can be modeled as a one-edge process which responds to *read* and *write* actions according to its current value. In particular, an edge t represents the state in which the variable is *true* while f is for *false*. A variable is attached to a location (a node), where actions are issued: r_t and w_t respectively for reading and writing *true*, and r_f, w_f for *false*. The behaviour of a variable attached to a node y, whose states are therefore represented by the graphs $t(y)$ and $f(y)$, is described by the following axioms:

$$
\begin{array}{ll}
t(y) \xrightarrow{y,w_f} f(y) & \qquad f(y) \xrightarrow{y,w_t} t(y) \\
t(y) \xrightarrow{y,w_t} t(y) & \qquad f(y) \xrightarrow{y,w_f} f(y) \\
t(y) \xrightarrow{y,r_t} t(y) & \qquad f(y) \xrightarrow{y,r_f} f(y)
\end{array}
$$

The following process $\mathit{if}(x\,y\,z_1 z_2)$ acts as an *if-then-else*: when stimulated by an action a on its node x, it tests the variable at y and, if it is true, moves to z_1 all possible processes attached to x, or otherwise it moves them to z_2.

$$
\mathit{if}(x\,y\,z_1 z_2) \xrightarrow{\substack{y,\overline{r}_t \\ x,a,z_1}} \mathit{if}(x\,y\,z_1 z_2) \qquad \mathit{if}(x\,y\,z_1 z_2) \xrightarrow{\substack{y,\overline{r}_f \\ x,a,z_2}} \mathit{if}(x\,y\,z_1 z_2)
$$

A process $p(x)$ approaches the *if-then-else* by issuing an $\overline{a}$ action at x. In doing so it passes a "parameter" u along the channel, to be unified with z_1 or z_2 during synchronisation: $p(x) \xrightarrow{x,\overline{a},u} p(u)$. Then, for example, the composite process $p(x) \,|\, \mathit{if}(x\,y\,z_1 z_2) \,|\, t(y)$ rewrites to $p(z_1) \,|\, \mathit{if}(x\,y\,z_1 z_2) \,|\, t(y)$ by a transition with label $\{(y, \overline{r}_t), (y, r_t), (x, \overline{a}, u), (x, a, z_1)\}$. The transition is depicted in Figure 1.

3 Comparison with SHR

Although a formal comparison of Synchronising Graphs with Synchronized Hyperedge Replacement (SHR [FMT01]) is beyond the scope of the present article, we still want to list the main differences between the two related

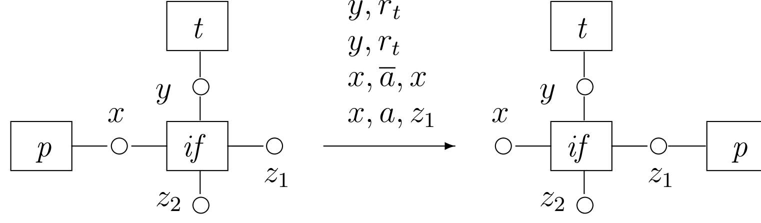


Figure 1. `if-then-else`.

systems. First we should notice that SHR adopts a syntactical presentation of graphs, defined as *judgments* of the form:

$$\Gamma \vdash G,$$

where Γ is a set of nodes and G is a *term* generated by the following grammar:

$$G \ ::= \ L(\mathbf{x}) \mid G|G \mid (\nu y)G|\, nil$$

(see the appendix for a short account of SHR, and [FMT01] for full detail). Structural equivalence must therefore be introduced to equate, for example, two graphs $\Gamma \vdash G|H$ and $\Gamma \vdash H|G$. The difference is just notational in SG, where graph are semantical objects. Besides this, we identify by means of three slogans the main features distinguishing SG from SHR.

1. Agents can quantum-leap.

That is, in SG $e(x) \xrightarrow{\emptyset} e(y)$ is a perfectly well defined transition, in which e leaps at node y from x without any further ado. There are two reasons why this is not allowed in SHR. First, no new node (like y in the example) is allowed to appear to the right hand side of a transition unless it is "declared" as argument of some action in Λ. Second, nodes cannot just be abandoned (like x): Consider an application in which agent $d(x\,y)$ dies. This is done in SG by a transition $d(x\,y) \xrightarrow{\emptyset} \emptyset$, which is mimicked in SHR by the production

$$x, y \vdash d(x\,y) \xrightarrow{\emptyset, id} x, y \vdash nil. \tag{1}$$

The above transition could occur, for example, in a larger context including an agent $e(y\,z)$. In SG: $d(x\,y) \mid e(y\,z) \xrightarrow{\emptyset} e(y\,z)$, while in SHR:

$$x, y, z \vdash d(x\,y) \mid e(y\,z) \xrightarrow{\emptyset, id} x, y, z \vdash e(y\,z).$$

Node x remains to the right hand side of the transition even if there are no edges attached to it (while x is just forgotten in SG). In such a case, SHR does not let e to proceed any further (computation stops), because the context Γ of a well-formed SHR production $\Gamma \vdash G \rightarrow [\ldots]$ must include *exactly* the free

nodes of G. Then, if computation is to proceed, d must fuse x and y at the act of dying, and (1) must be replaced by a less obvious

$$x, y \vdash d(x\,y) \xrightarrow{\emptyset,\{x \mapsto y\}} y \vdash nil\,.$$

This issue has been addressed in [HM01], where an additional *weakening* rule is introduced (in the context of an interleaving version of SHR).

2. Not all nodes are born equal.

While in SHR nodes are treated as variables, in SG they are thought of as *resources*, some of which providing capabilities which others may not have. For example an agent e may be able to rewrite to d when attached to node x but not if attached to y. This means that $e(x) \to d(x)$ can be axiom of a theory which does not allow the transition $e(y) \to d(y)$. On the other hand, the behaviour of an agent is specified in SHR by providing *productions* (that is, axiom schemes) the nodes of which can be freely instantiated. Hence, for example, the production $x \vdash e(x) \xrightarrow{(x,a),id} x \vdash d(x)$ automatically grants to agent e the rewrite $y \vdash e(y) \xrightarrow{(y,a),id} y \vdash d(y)$.

3. All nodes are born free.

Consider the transition $e(x) \xrightarrow{x,a,y} d(x\,y)$. Nodes such as y, that are arguments of actions labeling the transition but do not appear in the graph to the left, are treated in SG as *free* nodes (no restriction is applied to y in $d(x\,y)$). Then, as expected, synchronising the above transition with $e'(x) \xrightarrow{x,\bar{a},z} d'(x\,z)$, we obtain:

$$e(x)|e'(x) \xrightarrow{\substack{x,\bar{a},z \\ x,a,y}} d(x\,y)|d'(x\,y).$$

Conversely, SHR treats the nodes such as y above as *bound* names: synchronising $x \vdash e(x) \xrightarrow{(x,a,y),id} x, y \vdash d(x, y)$ with $x \vdash e'(x) \xrightarrow{(x,\bar{a},z),id} x, z \vdash d'(x, z)$ we obtain:

$$x \vdash e(x)|e'(x) \xrightarrow{\substack{(x,\bar{a},z),id \\ (x,a,y),id}} \nu\, y\, (d(x\,y)|d'(x\,y)).$$

4 S-graph rewriting

Let $\mathcal{F} = \{f, g, h \dots\}$ and $\mathcal{V} = \{x, y, z \dots\}$ be disjoint sets of *symbols*, called respectively *functions* and *variables*. The metavariable s will range over the set $\mathcal{F} \cup \mathcal{V}$ of symbols. We define *S-graphs* to be the synchronising graphs whose nodes are variables and whose edges are labeled by symbols. We use a bold $\mathbf{s}$ to denote an edge labeled by s. The *theory* of S-graphs features a

set $Act = \{f, \overline{f}, \dots x, \overline{x} \dots\}$ of actions, including all symbols and their complements. The axioms of the theory include the identities and all instances of the following axiom schemes (28 characters, including commas):

$$\mathbf{s}(x\,\boldsymbol{y}) \xrightarrow{x,s,\boldsymbol{y}} \mathbf{s}(x\,\boldsymbol{y})$$

$$\mathbf{s}(x\,\boldsymbol{y}) \xrightarrow{x,\overline{s},\boldsymbol{y}} \emptyset.$$

Proposition 1 *All computations originating at a finite S-graph terminate.*

By *terminate* we mean that any computation of S-graphs includes only a finite number of non-identity transitions. When a graph H is reached, from which no further transition is possible but the identity, H is called the *result* of the computation. The above proposition holds because each synchronisation reduces the number of edges by one.

Proposition 2 *The complexity of a computation originating at an S-graph with n tentacles is $n \cdot log^* n$.*

Proof. We say that a synchronisation $\{(x, s, \boldsymbol{y}), (x, \overline{s}, \boldsymbol{z})\}$ has length m when $m = |\boldsymbol{y}| = |\boldsymbol{z}|$. Each such synchronisation involves the unification of the lists $\boldsymbol{y}$ and $\boldsymbol{z}$ which, including the update of the graph, costs $m \cdot log^* m$ [BS01]. Each occurrence of a variable z_i in the list $\boldsymbol{z}$ corresponds to a tentacle of the graph attached to z_i, which disappears with the action $\overline{s}$. Then, the sum of the lengths of all synchronisations occurring during computation is bound by the total number of tentacles. □

Let T be the set of (*finite*) terms inductively defined over $\mathcal{F}$ and $\mathcal{V}$, that is the syntactic objects of the form:

$$t \;::=\; x \mid f(t_1, \dots t_n). \tag{2}$$

A family of functions $[\![\,_\,]\!]_x$ indexed by nodes translates terms into S-graphs:

$$[\![x]\!]_x = \mathbf{x}(x)$$

$$[\![f(t_1, \dots t_n)]\!]_x = \mathbf{f}(x\,y_1 \dots y_n) \mid [\![t_1]\!]_{y_1} \mid \dots \mid [\![t_n]\!]_{y_n},$$

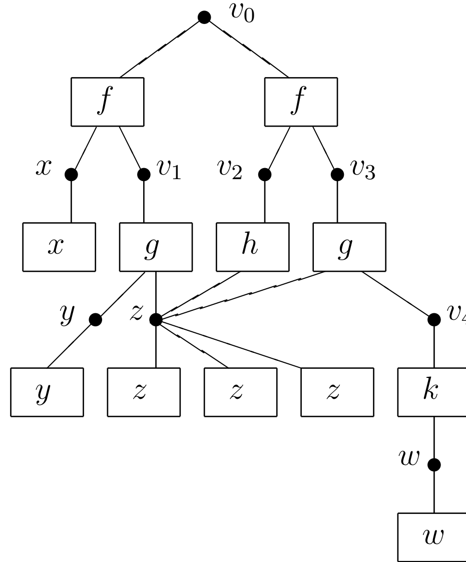
where $y_i = t_i$ when t_i is a variable, or otherwise y_i is a new node. The node to which the first tentacle of an edge is attached is called the *result node* of the edge. We define the S-graph $G_{t=t'}$ corresponding to an equation $t = t'$ between terms of T to be either $\mathbf{x}(x) \,|\, \mathbf{y}(x)$, when both $t = x$ and $t' = y$ are variables, or else

$$[\![t]\!]_x \mid [\![t']\!]_x,$$

where x is new if neither t nor t' are variables. The problem of unifying two terms t and t' is solved by a computation of $G_{t=t'}$. In order to avoid unsynchronised actions to occur in the computation, we further assume that all nodes in $G_{t=t'}$ are *restricted*. In Section 5 we prove that a computation of the S-graph $G_{t=t'}$ amounts to computing a most general unifier of t and t'. Before that, we provide intuition by developing an example.

Example.

Let t and t' be respectively the terms $f(x, g(y, z))$ and $f(h(z), g(z, k(w)))$. The graph $G_{t=t'}$ is depicted as follows.



A most general unifier of t and t' is the substitution mapping x to $h(k(w))$ and both y and z to $k(w)$. This substitution is represented (in a precise sense to be explained in Section 5) by the graph produced by the computation depicted in Figure 2, where

$$\Lambda = \{(z, z), (z, \overline{z}), (v_0, f, x\, v_1), (v_0, \overline{f}, v_2\, v_3)\} \text{ and}$$
$$\Theta = \{(z, z), (z, \overline{z}), (v_1, g, y\, z), (v_1, \overline{g}, z\, v_4)\}.$$

On the other hand, a computation originating at a graph $G_{t=t''}$, where t is as before and t'' is the term $f(h(z), k(w))$ would yield the graph depicted in Figure 3. This graph represents a *failure* condition, as it features the two "functional" edges, $\mathbf{g}(v_1\, y\, z)$ and $\mathbf{k}(v_1\, w)$, with $g \neq k$, that are attached by a common result node (v_1).

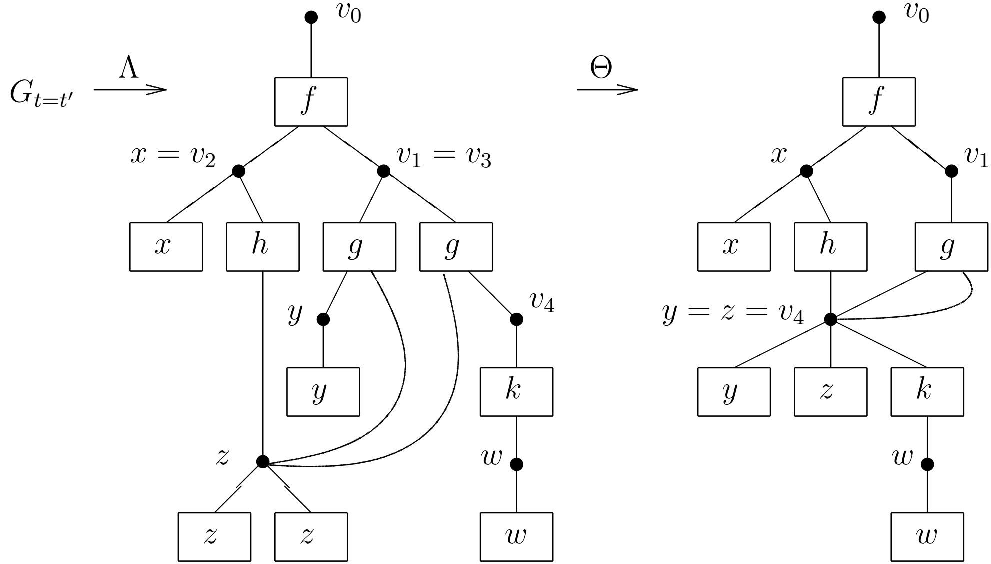


Figure 2. A computation of $G_{t=t'}$.

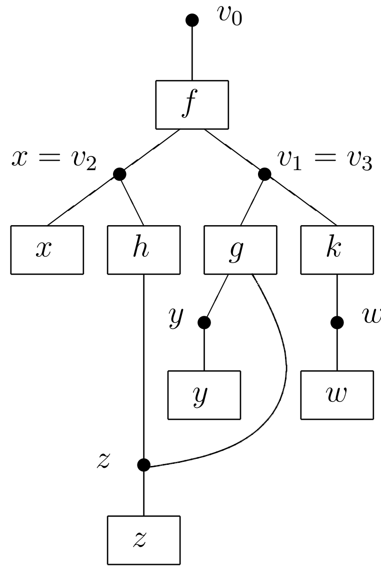
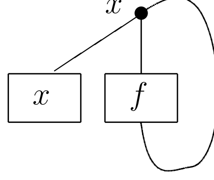


Figure 3. Failure in unifying $f(x, g(y, z))$ and $f(h(z), k(w))$

5 Unification of Rational Terms

A finite S-graphs may indeed represent an *infinite* (rational) term. This is indeed intended, as we want, for example, the unification of x and $f(x)$ to succeed. The unifier of these two terms is the substitution mapping x to the infinite term $f(f(f(\dots)))$, and is represented by the following graph,



which is the result of the one-step computation originating at $G_{x=f(x)}$. Therefore, let T_∞ be the set of possibly infinite terms co-inductively defined by the clauses in 2. There exists an isomorphism

$$\gamma : T_\infty \to \mathcal{V} \cup (\mathcal{F} \times \sum_n T_\infty^n),$$

which can be defined in categorical terms as the final coalgebra of the functor $F(X) = \mathcal{V} + (\mathcal{F} \times \sum_n X^n)$. We define *substitutions* to be *total* functions from $\mathcal{V}$ to T_∞. A substitution ρ is said to have *finite domain* if $\rho(x) \neq x$ only for finitely many x. Such a ρ is called *finite* if $\rho(x)$ is a finite term, for all x. Finite substitutions are finitely represented by giving their *interesting* (non-identity) equations only. For example, $\{x = f(y)\}$ represents the substitution $\{x = f(y), y = y, z = z, \dots\}$. As usual, the composite substitution $\rho \cdot \sigma$ maps x to $\rho(\sigma(x))$.

Given an S-graph G, we let $terms(G)$ be the largest $|G|$-indexed family of sets $terms_x(G) \subseteq T_\infty$ such that, if $s(t_1, \dots t_n) \in terms_x(G)$ then $\mathbf{s}(x, y_1, \dots y_n)$ is an edge of G and $t_i \in terms_{y_i}(G)$, for all i. Note that, by writing $s(t_1, \dots t_n)$ we mean to capture both function terms $f(\dots)$ and variables, in which case it is understood that n is 0. We say that G includes an equation $t_1 = t_2$ if $\{t_1, t_2\} \subseteq terms_x(G)$ for some $x \in |G|$. By a slight abuse, we say that an S-graph includes a substitution when it includes all the interesting equations in it.

Theorem 3 *Let $G_{t=t'} \xrightarrow{*} H$ be a computation terminating at H. One of the following conditions holds:*

- *i) H includes a subgraph $\mathbf{f}(x\,\boldsymbol{y}) \,|\, \mathbf{g}(x\,\boldsymbol{z})$ where either $f \neq g$ or $\boldsymbol{y}$ and $\boldsymbol{z}$ have different lengths. In such a case t and t' are not unifiable;*
- *ii) t and t' are unifiable and H includes a substitution which is a most general unifier of t and t'.*

When ii) holds, we say that the computation of $G_{t=t'}$ terminates *successfully*. Given a successful computation, the mgu included in H can be easily defined by exploiting the *finality* of $\gamma : T_\infty \to F(T_\infty)$, the property by which, for any coalgebra $g : A \to F(A)$, there exists a unique map $g^\dagger : A \to T_\infty$ such that

$$\gamma \circ g^\dagger = F(g^\dagger) \circ g. \tag{3}$$

Then, consider the following function $g : |H| \rightarrow F(|H|)$:

$$g(x) = \begin{cases} (f, \boldsymbol{y}) & \text{if } \mathbf{f}(x, \boldsymbol{y}) \text{ is an edge in } H, \\ x & \text{otherwise.} \end{cases}$$

This is a good definition because, when ii) holds, at most one $\mathcal{F}$-labeled edge is ever attached with its first tentacle to a single node. By spelling out equation (3) for the above g, we see that the coalgebra map $g^\dagger : |H| \rightarrow T_\infty$ thus obtained is such that, if an $\mathcal{F}$-labeled edge $\mathbf{f}(x, y_1, \dots y_n)$ exists in H, then $g^\dagger(x) = f(t_1 \dots t_n)$, with $t_i = g^\dagger(y_i)$ for all i, or else $g^\dagger(x) = x$. Below we show that the (finite domain) substitution mapping each variable $y \in \mathit{terms}_x(H)$ to $g^\dagger(x)$ is an mgu.

Lemma 4 *Let $G \rightarrow H$ be derivable in the theory of S-graphs. A substitution ρ unifies all equations in G if and only if it unifies all equations in H. Moreover, ρ is an mgu for G if and only if it is an mgu for H.*

The *if* direction of the first statement is trivial, as the equations of G are included in the equations of H. The *only if* holds because any equation of H which is not in G must be of the form $t_i = t'_i$, with $f(\dots t_i, \dots) = f(\dots t'_i, \dots)$ an equation of G. The second statement of the lemma follows easily from the first. The next lemma follows from the observation that t belongs to $\mathit{terms}_x([\![t]\!]_x)$, for t finite.

Lemma 5 *For any two finite terms t and t', the graph $G_{t=t'}$ includes the equation $t = t'$.*

Proof of theorem 3. Let H include no subgraph as in i), and let ρ be the substitution mapping each variable $y \in \mathit{terms}_x(H)$ to $g^\dagger(x)$, where g is as above. No variable w such that $\rho(w) \neq w$ ever appears in a term resulting from the application of ρ. Therefore ρ solves all equations in H and moreover it is most general. Hence, since $t = t'$ is the *unique* equation in $G_{t=t'}$, it follows from the lemmas that ρ is an mgu of t and t', as required. Otherwise, if H does include a subgraph as in i), again by the lemmas, any unifier t and t' would have to solve an equation $f(t_1, \dots t_n) = g(t'_1, \dots t'_m)$, where either $f \neq g$ or $n \neq m$, which cannot be. So t and t' are not unifiable. □

Proposition 6 *Let $G_{t=t'} \xrightarrow{*} H$ be a computation terminating successfully at H. The equation has a* finite *unifier if and only if H has no cycles.*

Proof. Observing that $t = t'$ has a finite solution if and only if it has a finite mgu, the result follows by the spelling of equation (3). □

It is folklore that there exist equations whose mgu is exponential in size with respect to the initial system. Therefore, in order to claim efficiency, we must split the problem of *producing* an mgu from its graph representation H

into two steps. The first step $H \mapsto \varsigma$ is performed in *linear time*, and produces a system of replacements $\varsigma = \{x_i = t_i\}_{i=1\ldots n}$ to be carried out sequentially. We call ς the *linear presentation* of an mgu. Notice that, when linearity is claimed in literature, e.g. in [PW76,MM82], it is always up to here. In the second step the n equations of ς are composed, thus producing a substitution ρ, which is an mgu in case H has no cycles. Otherwise an mgu is obtained by infinite unfolding of ρ.

The following program visits the result graph H of a computation of $G_{t=t'}$, detecting a possible failure condition, in the absence of which it prints a linear presentation of the mgu of t and t'. (It is arguable whether the program should be counted as part of our unification algorithm, in which unfortunate case the claimed 28 characters would raise to 332.) A global data structure is assumed, keeping track of which nodes have already been visited. All nodes are initially not visited. The algorithm consists in applying the following procedure `P` to the nodes of H until all nodes are visited. Given a node $x \in |H|$, a function `choose` picks an element from the set of variables y such that $y(x)$ is in H. If no such variable exists `choose` returns `null`.

```
term P (node x) is
   term result = choose(x);
   if x is visited return result;
   mark x as visited;
   if f(x,y1,... yn) and g(x,z1,... zm) are in H
      and (f != g or m != n)
      { print FAIL;
        stop;
      }
   if f(x,y1,... yn) is in H
      { for i=1...n do ti := P(yi);
        if result = null return f(t1,...tn);
        else print result = f(t1,...tn);
      }
   for all z(x) in H do
       if (not z = result) print z = result;
   return result;
```

Note that, when `x` is visited, `result` is always different from `null` at line 3. In fact, this is true if `P` were to be applied to $G_{t=t'}$, and the property is preserved by reduction. Although the one but last line looks funny, it is so intended: if `y` is chosen to replace all variables that are attached to node `x`, an equation `z = y` is to be printed for all such variables `z`, except for `y`.

The complexity of `P` is clearly linear in the size of H. As for correctness, it is routine to show that, if $\{x_i = t_i\}_{i=1\ldots n}$ is the result of the above algorithm, the finite substitution $\rho = \{x_1 = t_1\} \cdot \ldots (\{x_{n-1} = t_{n-1}\} \cdot \{x_n = t_n\})$, once

restricted to its non-identity equations, is in *Greibach form*, that is: it includes no equation of the form $x_i = x_j$, with x_j in the domain of ρ. It is well-known that such a system has a unique solution [Cou83]. It is also easy to show that, when H has no cycles, ρ is in *solved* form, that is: no variable in its domain ever appears in any of the t_i, in which case ρ itself is the solution.

6 Conclusions

Using synchronising graphs we provided a compact, declarative solution to the problem of syntactic unification of rational terms. While matching the best algorithms proposed in literature in terms of complexity, the proposed solution is meant to show that the simplicity of the model is not obtained at the cost of expressive power. Moreover, our approach can be easily extended to capture (first order) *equational* unification, as in [Plu99], a rather large field of applications where synchronising graphs could be employed as semantic framework for writing executable specifications.

References

[BS01] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.

[CG00] L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 1(240):177–213, 2000.

[Cou80] G. Cousineau. An algebraic definition for control structures. *Theoretical Computer Science*, 12(2):175–192, October 1980.

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, March 1983.

[CT04] P. Cenciarelli and A. Tiberi. Synchronising Graphs. Submitted, 2004.

[CTT04] P. Cenciarelli, I. Talamo, and A. Tiberi. Ambient Graph Rewriting. Proceedings of WRLA'04. To appear in Elsevier ENTCS, 2004.

[DM87] P. Degano and U. Montanari. A model of distributrd systems based on graph rewriting. *Journal of the ACM*, 34:411–449, 1987.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2*. World Scientific, Singapore, 1999.

[FMT01] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. *Proc.ITCS 01, Springer LNCS 2202*, 2001.

[HM01] Hirsch and Montanari. Synchronized hyperedge replacement with name mobility. In *CONCUR: 12th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2001.

[Hue76] G. Huet. *Résolution d'équations dans les langages d'ordre 1, 2, . . . , ω.* PhD thesis, Université Paris, 7, 1976.

[Jaf84] J. Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2(3), 1984.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[Plu99] D. Plump. Term graph rewriting. In Ehrig et al. [EEKR99], pages 3–61.

[Pro95] Programming Systems Group, Swedish Institute of Computer Science, Sweden. *SICStus Prolog User's Manual*, 1995.

[PV98] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. *Proc. LICS'98*, 1998.

[PW76] M. S. Paterson and M. N. Wegman. Linear unification. In *Proc. of 8th ACM Symp. on Theory of Computing*, pages 181–186. ACM Press, 1976.

[RH01] J. Riely and M. Hennessy. Distributed Processes and Location Failures. *Theoretical Computer Science*, 266:693–735, 2001.

[Rob65] J. A. Robinson. A machine-oriented logic based on resolution principle. *Journal of the ACM*, 12(1):23–49, January 1965.

Appendix: Synchronized Hyperedge Replacement

The material in this appendix comes from [FMT01], to which we refer for more detail. In *SHR* a transition is a *logical sequent* of the following form:

$$\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \phi \vdash G_2,$$

where Λ is a partial function on Γ which maps nodes to pairs $(action, nodes)$, and $\pi : \Gamma \to \Gamma$ is a total function called *fusion substitution*, which allows nodes to be "fused." The function Λ is similar to the corresponding relation in *SG* while π is used to keep track of the unifications induced by synchronisation. The inference rules are:

$$(\textsc{par})\ \frac{\Gamma_1 \vdash G_1 \xrightarrow{\Lambda,\pi} \phi_1 \vdash G_2 \qquad \Gamma_2 \vdash G' \xrightarrow{\Lambda',\pi'} \phi_2 \vdash G''}{\Gamma_1, \Gamma_2 \vdash G_1|G' \xrightarrow{\Lambda\cup\Lambda',\pi\cup\pi'} \phi_1, \phi_2 \vdash G_2|G''}\ (\psi_1)$$

$$(\textsc{merge})\ \frac{\Gamma \vdash G \xrightarrow{\Lambda,\pi} \phi \vdash G'}{\sigma\Gamma \vdash \sigma G \xrightarrow{\Lambda',\pi'} \phi' \vdash \nu\mathbf{u}.\rho(\sigma(G'))}\ (\psi_2)$$

$$(\textsc{res})\ \frac{\Gamma, x \vdash G \xrightarrow{\Lambda,\pi} \phi \vdash G'}{\Gamma \vdash \nu x.G \xrightarrow{\Lambda',\pi|\Gamma} \phi' \vdash \nu\mathbf{z}.G'}\ (\psi_3)$$

$$\psi_1 = (\Gamma_1 \cup \phi_1) \cap (\Gamma_2 \cup \phi_2) = \emptyset$$

$$\psi_2 = \begin{cases} \Delta \cap \sigma(\Gamma) = \emptyset \text{ and } \forall x \in \Delta.\sigma(x) = x \\ \sigma(x) = \sigma(y) \wedge \Lambda(x)\downarrow \wedge \Lambda(y)\downarrow \wedge x \neq y \implies \\ \quad (\forall z \notin \{x, y\}).\sigma(z) = \sigma(x) \implies \Lambda(z)\uparrow) \\ \quad \wedge\ \Lambda(x) = (a, \mathbf{v}) \wedge \Lambda(y) = (\bar{a}, \mathbf{w}) \wedge a \neq \tau \\ \rho = \mathbf{mgu}\{\sigma(\mathbf{v}) = \sigma(\mathbf{u}) \mid \sigma(x) = \sigma(y) \wedge \Lambda(x) = (a, \mathbf{u}) \wedge \Lambda(y) = (\bar{a}, \mathbf{v})\} \\ \Lambda'(z) = \begin{cases} (\tau, \langle\rangle), & if\ \sigma(x) = \sigma(y) = z \wedge x \neq y \wedge \Lambda(x)\downarrow \wedge \Lambda(y)\downarrow \\ \rho(\sigma(\Lambda))(z), & otherwise \end{cases} \\ \pi'(\sigma(x)) = \rho(\sigma(\pi(x))) \\ \mathbf{u} = (\rho(\sigma(\phi)) \setminus \phi' \end{cases}$$

$$\psi_3 = \begin{cases} (\pi(x) = \pi(y) \wedge x \neq u) \implies \pi(x) \neq x \\ \Lambda(x)\uparrow\ or\ \Lambda(x) = (\tau, \langle\rangle) \\ \Lambda' = \Lambda \setminus \{(x, \tau, \langle\rangle\} \\ \mathbf{z} = \phi \setminus \phi' \end{cases}$$

where $n(\Lambda) = \{x \mid (a, \mathbf{u}x\mathbf{v}) \in \Lambda\}$ and $\Delta = n(\Lambda) \setminus \Gamma$