Semantics-Preserving DPO-Based Term Graph Rewriting[*]

Wolfram Kahl
McMaster University, Hamilton, Ontario, Canada,
kahl@cas.mcmaster.ca

Yuhang Zhao
McMaster University, Hamilton, Ontario, Canada,
zhaoy36@mcmaster.ca

Term graph rewriting is important as "conceptual implementation" of the execution of functional programs, and of data-flow optimisations in compilers. One way to define term graph transformation rule application is via the well-established and intuitively accessible double-pushout (DPO) approach; we present a new result proving semantics preservation for such DPO-based term graph rewriting.

1 Introduction and Related Work

Term graph rewriting goes back to Wadsworth [Wad71], who proposed it as an efficient implementation mechanism for the λ-calculus. This aspect has remained dominant in the term graph literature; for example, Rose [Ros93] defines an operational semantics of a lazy functional programming language via term graph rewriting; Ariola, Klop and Plump [AKP00] study confluence of term graph rewriting using bisimilarity. When justifying term graph rewriting as a correct implementation technique (for, in particular, functional programming), most of the literature approaches this from the relationship with term rewriting. For example, when Plump [Plu02] writes about "Essentials of Term Graph Rewriting", soundness and completeness are considered only with respect to term rewriting. Kennaway *et al.* [KKSV93, KKSV94] define a notion of simulation to prove adequacy of term graph rewriting for finite and rational term rewriting.

When attempting to employ traditional categorial approaches to graph rewriting, the so-called "algebraic approach", to term graph rewriting, two main problems arise: First, categories of "standard" term graph homomorphisms typically do not have all pushouts, since unification translates into pushouts, and second, the interface graphs needed both for the double-pushout (DPO) approach and for the single-pushout approach (to capture the domain of morphisms) are typically not term graphs, but some kind of "term graphs with holes". Term graph rewriting is therefore a niche of graph transformation that has pioneered exploration of formalisms where pushout squares are generalised in some way, in particular by using different morphisms in the horizontal and vertical directions of the standard DPO drawing.

For example, Banach [Ban93] defines "DACTL" term graph rewriting using a modified opfibration, and Kahl [Kah96, Kah97] uses both fibrations and opfibrations to define rewriting of term graphs with variable binding. A different approach to using separate classes of horizontal and vertical morphisms for term graph rewriting has been proposed by Duval *et al.* [DEP09], who are using a specific rule concept as morphisms in the horizontal direction in their "heterogeneous pushout approach". More recently, motivated by attributed graphs, which share some characteristics with term graphs, Habel and Plump [HP12] propose "$\mathcal{M},\mathcal{N}$-adhesive transformation systems" as one general framework to accommodate different classes of morphisms in the horizontal and vertical directions of the double-pushout setting.

Corradini and Gadducci [CG99a, CG02] opened up a new way of investigating term graphs by defining gs-monoidal categories as a variant of Lawvere theories [Law63]. Gs-monoidal categories are

[*]This research is supported by the National Science and Engineering Research Council of Canada, NSERC.

an intermediate concept between symmetric monoidal categories and cartesian (monoidal) categories; the only difference with the latter is that, the "duplicator" transformation ∇ producing diagonal maps $\nabla_A : A \to A \otimes A$ and the "terminator" transformation $!$ with components $!_A : A \to \mathbb{1}$ are both *not* assumed to be natural transformations (that is, for a morphism $F : A \to B$, the equations $F \mathbin{⨾} \nabla_B = \nabla_A \mathbin{⨾} (F \otimes F)$ and $F \mathbin{⨾} !_B = !_A$ do *not* necessarily hold.).

Corradini and Gadducci demonstrate in [CG99a] that taking natural numbers as objects and term graphs with m inputs and n outputs as morphisms from object m to object n produces a free gs-monoidal category, and thus they automatically obtain a functorial semantics for term graphs in arbitrary gs-monoidal categories, which include all Cartesian categories, and so in particular also Set. Continuing this line of work, Corradini and Gadducci obtain semantics preservation for a low-level definition of "ranked dag rewriting" and involving "contexts" analogous to the contexts of term rewriting [CG97, CG99b]. Finally, in [CG05] they show a quasi-adhesive category of term graphs, but emphasise that adhesive categorial rewriting in that category does not quite match term graph rewriting. They mention in their conclusion that a possible alternative is to perform the DPO on a super-category of hypergraphs; this is essentially the approach we are elaborating here. As an example consider Fig. 1, showing the application of a rule corresponding to the term rule $(x_1 + x_2) - x_2 \longrightarrow x_1$ to rewrite a term graph corresponding to $y_1 + ((y_2 + y_3) - y_3) \times y_4$ to $y_1 + y_2 \times y_4$.

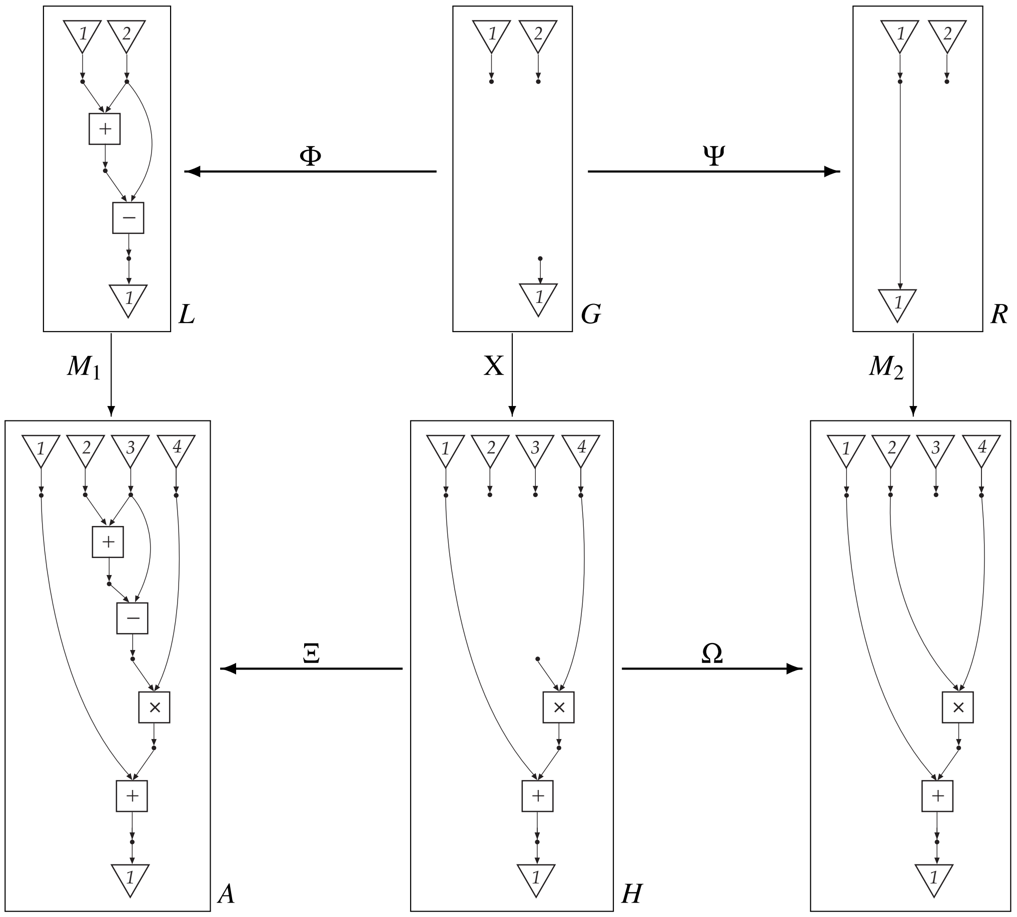


Figure 1: Example term graph rewriting step

In Sect. 2 we provide details about term graphs and how we draw them, and the definition of gs-monoidal categories with explanations how term graphs populate that concept. In Sect. 3 we present the adaptations we use to obtain a DPO-based definition of term graph transformation, and in Sect. 4 we sketch the proof that such transformation steps are semantics-preserving if the rule sides are semantically equivalent.

2 Background: Term Graphs and GS-Monoidal Categories

We are using the "jungle" view of term graphs, which goes back to Hoffmann and Plump [HP91] and Corradini and Rossi [CR93], since this is the view used by the gs-monoidal semantics, where nodes translate into objects and (hyper-)edges into morphisms.

We assume a set $\mathcal{L}$ of *edge labels* together with an function $\mathsf{arity} : \mathcal{L} \to \mathbb{N}$ prescribing for each label the number of inputs the corresponding edges take. We write $\mathsf{Fin}_k := \{i : \mathbb{N} \mid i < k\}$ for the set containing the first k natural numbers, and will use this in particular for the set of graph input nodes.

Definition 2.1. The set of *directed hypergraph graphs with m inputs and n outputs* will be denoted by $\mathsf{DHG}_{m,n}$. An element of $\mathsf{DHG}_{m,n}$ is a tuple $(\mathcal{I}, \mathcal{E}, \mathsf{eLabel}, \mathsf{eOut}, \mathsf{eIn}, \mathsf{gOut})$ consisting of two sets,

- a set $\mathcal{I}$ of *inner nodes*, from which we construct the set $\mathcal{N} = \mathsf{Fin}_m \uplus \mathcal{I}$ of *nodes* as disjoint union of the set Fin_m of *graph input nodes* and the set $\mathcal{I}$ of inner nodes,
- a set $\mathcal{E}$ of *(hyper-)edges*,

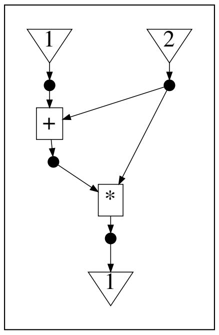
and four functions,

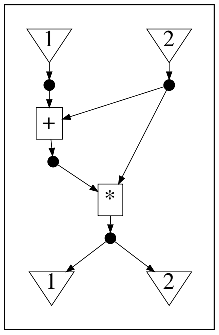
- $\mathsf{eLabel} : \mathcal{E} \to \mathcal{L}$ assigning each edge a label,
- $\mathsf{eOut} : \mathcal{E} \to \mathcal{I}$ assigning each edge a single *edge output node*, which has to be an inner node,
- $\mathsf{eIn} : \mathcal{E} \to \mathcal{N}^*$ assigning each edge a sequence of *edge input nodes*, which needs to have as its length the arity of the edge's label, that is, $\forall e : \mathcal{E} \bullet \mathsf{arity}(\mathsf{eLabel}(e)) = \mathsf{length}(\mathsf{eIn}(e))$, and
- $\mathsf{gOut} : \mathsf{Fin}_n \to \mathcal{N}$ assigning each output position a node.

A *term graph* is an acyclic directed hypergraph where eOut is bijective; we write $\mathsf{TG}_{m,n}$ for the set of term graphs with m inputs and n outputs. □

When drawing such hypergraphs and term graphs, we start with the inputs on top and proceed down to the outputs, drawing nodes as bullets, and (hyper-)edges as labelled boxes connected to nodes via (implicitly ordered) input-tentacles and exactly one output-tentacle. (Although edges with multiple outputs have uses for example in the code graphs of [KAC06, AK09], most of the literature, including all the cited work by Corradini and Gadducci, only considers single-output operations (edges), so we also do this here.) Graph input nodes are declared by attaching a triangle pointing to the input node — input nodes are necessarily distinct, and cannot be output nodes of edges. Graph input nodes are frequently called "variable nodes", and translated into distinct variables for a term reading. Graph output nodes (in the literature frequently referred to as "roots") are declared by attaching a triangle pointing away from them — any node can be used as a graph output any number of times.

A graph with multiple graph outputs is interpreted as standing for a tuple of terms: The left box in the following drawing depicts a term graph (from $\mathsf{TG}_{2,1}$) corresponding to the term "$(x_1 + x_2) * x_2$", while the term graph (from $\mathsf{TG}_{2,2}$) in the right box corresponds to the pair of terms "$((x_1 + x_2) * x_2, (x_1 + x_2) * x_2)$" (or, if **let**-definitions are available, "**let** $z = (x_1 + x_2) * x_2$ **in** (z, z)"):





Term graphs with sequential composition ($⨾$) and parallel composition ($\otimes$) form a gs-monoidal category according to Corradini and Gadducci [CG99a]: The objects are the natural numbers (interpreted as numbers of nodes in the graph input interface, respectively graph output interface), and term graphs with m inputs and n outputs are morphisms from m to n.

Definition 2.2. For a *category* $(\mathsf{Obj}, \mathsf{Mor}, \mathsf{src}, \mathsf{trg}, \mathbb{I}, ⨾)$, we write $f : \mathcal{A} \to \mathcal{B}$ instead of $\mathsf{src}(f) = \mathcal{A} \wedge \mathsf{trg}(f) = \mathcal{B}$; composition of two morphisms $f : \mathcal{A} \to \mathcal{B}$ and $g : \mathcal{B} \to \mathcal{C}$ is written "$f ⨾ g$", and the identity for object $\mathcal{A}$ is $\mathbb{I}_{\mathcal{A}}$.

A *symmetric strict monoidal category* [ML71] $(\mathbf{C}_0, \otimes, \mathbb{1}, \mathbb{X})$ consists of a category $\mathbf{C}_0$, a strictly associative monoidal bifunctor $\otimes$ with $\mathbb{1}$ as its strict unit, and a transformation $\mathbb{X}$ that associates with every two objects $\mathcal{A}$ and $\mathcal{B}$ an arrow $\mathbb{X}_{\mathcal{A},\mathcal{B}} : \mathcal{A} \otimes \mathcal{B} \to \mathcal{B} \otimes \mathcal{A}$ with $\mathbb{X}_{\mathbb{1},\mathbb{1}} = \mathbb{I}_{\mathbb{1}}$ and:

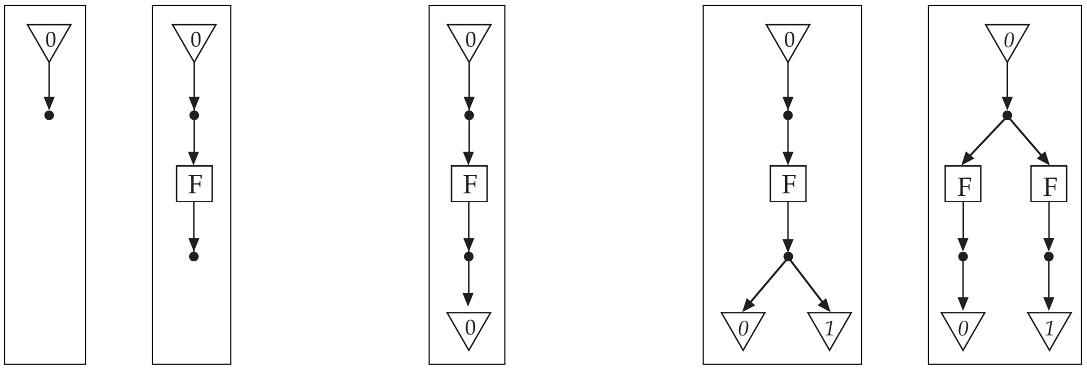
$$(F \otimes G) ⨾ \mathbb{X}_{\mathcal{C},\mathcal{D}} = \mathbb{X}_{\mathcal{A},\mathcal{B}} ⨾ (G \otimes F) \quad , \quad \mathbb{X}_{\mathcal{A},\mathcal{B}} ⨾ \mathbb{X}_{\mathcal{B},\mathcal{A}} = \mathbb{I}_{\mathcal{A}} \otimes \mathbb{I}_{\mathcal{B}} \quad , \quad \mathbb{X}_{\mathcal{A}\otimes\mathcal{B},\mathcal{C}} = (\mathbb{I}_{\mathcal{A}} \otimes \mathbb{X}_{\mathcal{B},\mathcal{C}}) ⨾ (\mathbb{X}_{\mathcal{A},\mathcal{C}} \otimes \mathbb{I}_{\mathcal{B}}) \ .$$

$(\mathbf{C}_0, \otimes, \mathbb{1}, \mathbb{X}, \nabla, !)$ is a *strict gs-monoidal category* iff $(\mathbf{C}_0, \otimes, \mathbb{1}, \mathbb{X})$ is a symmetric strict monoidal category, and

- $!$ associates with every object $\mathcal{A}$ of $\mathbf{C}_0$ an arrow $!_{\mathcal{A}} : \mathcal{A} \to \mathbb{1}$, and
- ∇ associates with every object $\mathcal{A}$ of $\mathbf{C}_0$ an arrow $\nabla_{\mathcal{A}} : \mathcal{A} \to \mathcal{A} \otimes \mathcal{A}$, such that:

$$\begin{array}{lll} \nabla_{\mathcal{A}} ⨾ (\mathbb{I}_{\mathcal{A}} \otimes \nabla_{\mathcal{A}}) = \nabla_{\mathcal{A}} ⨾ (\nabla_{\mathcal{A}} \otimes \mathbb{I}_{\mathcal{A}}) & \nabla_{\mathcal{A}} ⨾ \mathbb{X}_{\mathcal{A},\mathcal{A}} = \nabla_{\mathcal{A}} & \nabla_{\mathcal{A}} ⨾ (\mathbb{I}_{\mathcal{A}} \otimes !_{\mathcal{A}}) = \mathbb{I}_{\mathcal{A}} \\ \nabla_{\mathcal{A}\otimes\mathcal{B}} ⨾ (\mathbb{I}_{\mathcal{A}} \otimes \mathbb{X}_{\mathcal{B},\mathcal{A}} \otimes \mathbb{I}_{\mathcal{B}}) = \nabla_{\mathcal{A}} \otimes \nabla_{\mathcal{B}} & !_{\mathcal{A}\otimes\mathcal{B}} = !_{\mathcal{A}} \otimes !_{\mathcal{B}} & \mathbb{I}_{\mathbb{1}} = !_{\mathbb{1}} = \nabla_{\mathbb{1}} \end{array} \qquad \square$$

For term graphs, the lack of naturality of the "terminator" transformation $!$ means that *garbage* (nodes from which no output is reachable) makes a difference, such as between the two graphs to the left below, and the lack of naturality of the "duplicator" transformation ∇ means that *sharing* (use of nodes in more than one consumer rôle, that is, as inputs for edges or as graph outputs) makes a difference, such as between the two graphs to the right below. (The words "garbage" and "sharing" motivate the name "gs-monoidal".)



$!_A$ $\quad F ⨾ !_B$ $\quad F$ $\quad F ⨾ \nabla_B$ $\quad \nabla_A ⨾ (F \otimes F)$

Corradini and Gadducci [CG99a] show furthermore that the term graphs over a given signature are arrows of the gs-monoidal category freely generated by that signature; therefore, there always exists a unique functor from the gs-monoidal category of term graphs to any gs-monoidal category. This induces a functorial semantics for term graphs in any gs-monoidal category. (This will frequently be some (cartesian) category of sets, with some set $\mathcal{V}$ chosen as set of *values* "at a node"; a term graph with m inputs and n outputs then has a function of type $\mathcal{V}^m \to \mathcal{V}^n$ as semantics. For code generation applications, one may construct non-cartesian gs-monoidal semantics categories where morphisms contain information about resource usage, such as number of instructions.)

3 Adapted DPO for Term Graph Rewriting

We will use the naming of graphs and morphisms used in Fig. 2 for double-square diagrams in the shape of double pushouts.

$$\begin{tikzcd}
L \arrow[d, "M_1"'] & G \arrow[l, "\Phi"'] \arrow[r, "\Psi"] \arrow[d, "X"'] & R \arrow[d, "M_2"] \\
A & H \arrow[l, "\Xi"'] \arrow[r, "\Omega"] & B
\end{tikzcd}$$

Figure 2: Naming of objects and morphism in "DPO-shape" diagrams

The example term graph transformation step in our adapted DPO approach shown in Fig. 1 in the introduction in effect closely corresponds to the more low-level definitions of term graph transformation dominant in the literature: the "host graph" (or "context graph") H can be thought of as obtained from the "application graph" A by deleting all edges and inner nodes of A which have a pre-image in L, but no pre-image (via $\Phi \fatsemi M_1$) in G, and the "result graph" B is obtained from H by "gluing in" the right-hand side R.

The gluing graph G and the host graph H are obviously not jungles, since they have nodes that are neither graph input nodes nor edge output nodes, but they still are directed hypergraphs (DHGs) in the sense of Def. 2.1.

Both for DHGs and for term graphs we distinguish *matchings*, which preserve edge labelling and incidence structure, from *homomorphisms*, which in addition preserve also graph input and output structure:

Definition 3.1. A *DHG matching* $\Phi = (\Phi_{\mathcal{N}}, \Phi_{\mathcal{E}})$ from $G_1 : \mathsf{DHG}_{m_1,n_1}$ to $G_2 : \mathsf{DHG}_{m_2,n_2}$ consists of two functions $\Phi_{\mathcal{N}} : \mathcal{N}_1 \to \mathcal{N}_2$ and $\Phi_{\mathcal{E}} : \mathcal{E}_1 \to \mathcal{E}_2$ satisfying:

$$\mathsf{eOut}_2 \circ \Phi_{\mathcal{E}} = \Phi_{\mathcal{N}} \circ \mathsf{eOut}_1 \;, \qquad \mathsf{eLabel}_2 \circ \Phi_{\mathcal{E}} = \mathsf{eLabel}_1 \;, \qquad \text{and} \qquad \mathsf{eIn}_2 \circ \Phi_{\mathcal{E}} = \mathsf{map}\; \Phi_{\mathcal{N}} \circ \mathsf{eIn}_1 \,.$$

A *DHG homomorphism* $\Phi = (\Phi_{\mathcal{I}}, \Phi_{\mathcal{E}})$ from $G_1 : \mathsf{DHG}_{m,n}$ to $G_2 : \mathsf{DHG}_{m,n}$ consists of two functions $\Phi_{\mathcal{I}} : \mathcal{I}_1 \to \mathcal{I}_2$ and $\Phi_{\mathcal{E}} : \mathcal{E}_1 \to \mathcal{E}_2$ such that defining $\Phi_{\mathcal{N}} := Id_{\mathsf{Fin}_m} \uplus \Phi_{\mathcal{I}}$ turns $(\Phi_{\mathcal{N}}, \Phi_{\mathcal{E}})$ into a matching from G_1 to G_2 and additionally satisfies $\mathsf{gOut}_2 = \Phi_{\mathcal{N}} \circ \mathsf{gOut}_1$.

If G_1 and G_2 are term graphs, then a matching (respectively homomorphism) Φ from G_1 to G_2 is called a *term graph matching* (respectively *term graph homomorphism*). □

The diagram in Fig. 1 is then a double pushout in the category of DHG matchings, satisfying the following additional requirements:

Definition 3.2. A DPO diagram in the category of DHG matchings of the shape of Fig. 2 is called a *TG-DPO* iff:

- M_1 and M_2 are term graph matchings (which implies that L, R, A, and B all are term graphs),
- Φ, Ψ, Ξ, Ω are DHG homomorphisms. □

Superficially, this arrangement looks similar to that of the $\mathcal{M},\mathcal{N}$-adhesive categories of Habel and Plump [HP12] — we would use DHG homomorphisms for $\mathcal{M}$ and term graph matchings for $\mathcal{N}$. However, several of the conditions of $\mathcal{M},\mathcal{N}$-adhesive categories fail to hold for this setting.

The existence of a pushout complement in the category of DHG matchings is subject to the gluing condition as usual — both dangling and identification conflicts can occur.

If the rule $L \xleftarrow{\Phi} G \xrightarrow{\Psi} R$ consists of DHG homomorphisms, both the pushout complement construction for the left square and the pushout construction for the right square will yield DHG matchings Ξ and Ω that also respect the graph interface, and therefore are DHG homomorphisms.

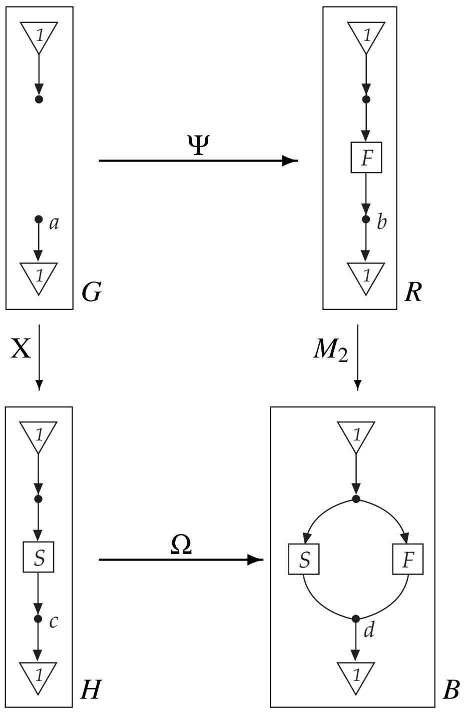


Figure 3: RHS edge conflict

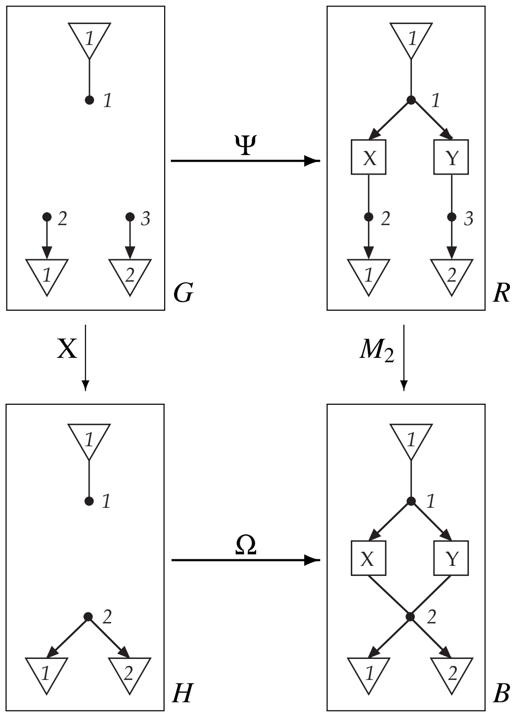


Figure 4: Non-injective host matching

For the right square of the DPO diagram, we finally have to ensure that B is a term graph, which is not trivial. First, the situation shown in Fig. 3 would lead to B not being a term graph — however, since the Φ-image of node a in L has to be either an input node or the output of an edge, such a situation cannot occur at least when the rule LHS Φ is injective. (If the image of a is an input node, then, with Φ preserving the graph interface, it cannot be injective. If the image of a is the output node of an edge in L, then the image in A of that edge needs to be also the image of the S-edge in H, which contradicts the left-hand pushout.) Second, also the example DHG matching pushout in Fig. 4 fails to produce a term graph B — this situation can be avoided by restricting the matching M_1 to be injective. (In effect, both constraints together correspond to the restriction to the "regular monos" of [CG05, Prop. 4.3].)

Since the right-hand side Ψ of the rule is a DHG homomorphism, it is automatically injective on input nodes; non-injectivity of Ψ therefore can only force identifications that are also "permissible" for the host graph, so we do not need to restrict Ψ to be injective, which would be highly unwelcome for term graph rewriting.

Therefore, DPOs in the DHG matching category can be used to rewrite term graphs with rules with injective left-hand sides, using only injective matchings (which takes care of the identification part of the gluing condition):

Theorem 3.3. Given a term graph rewriting rule $L \xleftarrow{\Phi} G \xrightarrow{\Psi} R$ where L and R are term graphs and Φ and Ψ are DHG homomorphisms, with Φ injective, and given further an injective term graph matching $L \xrightarrow{M_1} A$, then this setting can be completed to a TG-DPO if the dangling condition holds for M_1. □

The fact that Φ is injective implies that the output nodes of L are disjoint from the input nodes; we call such a term graph *solid*.

4 Semantics Preservation of DPO-Transformation of Term Graphs

While the fact that term graphs form a free gs-monoidal category gives us semantics of term graphs, it does not give us semantics of DHGs such as the gluing and host graphs in most typical rewriting steps. Rather than trying to artificially obtain some semantics for DHGs "with holes", we will transfer the necessary information "across the host graph H" at the DHG level.

A starting point could be the decomposition of term graphs into gs-monoidal expressions as described in [CG99a]. However, instead of extending this expression type into a type of contexts by including "placeholders" as proposed in [CG02], we define contexts at the level of graphs:

Definition 4.1. An *m,n-context (k,A_1,A_2) for an i,j-parameter* consists of:

- an *internal interface* object k,
- a *top part* term graph $A_1 : \mathsf{TG}_{m,i+k}$, and
- a *bottom part* term graph $A_2 : \mathsf{TG}_{j+k,n}$. □

In the following, we continue to use "$\fatsemi$" as sequential composition operator for term graphs, and "$\otimes$" for parallel composition. Furthermore, "$\mathbb{I}_k$" denotes the *identity* term graph with k inputs that are also its outputs, in the same sequence. The empty DHG with i inputs, and with j distinct output nodes that are disjoint from the input nodes is written "$\perp_{i,j}$"; for the sub-category of DHG *homomorphisms* restricted to DHGs with i inputs and j outputs, $\perp_{i,j}$ is the initial object.

Definition 4.2. An m,n-context (k,A_1,A_2) for an i,j-parameter is called an *image context for* an injective term graph matching $M_1 : L \to A$ starting from term graph $L : \mathsf{TG}_{i,j}$ iff $A \cong A_1 \fatsemi (L \otimes \mathbb{I}_k) \fatsemi A_2$ and the nodes and edges of L in that expression precisely constitute the image of M_1 in A. □

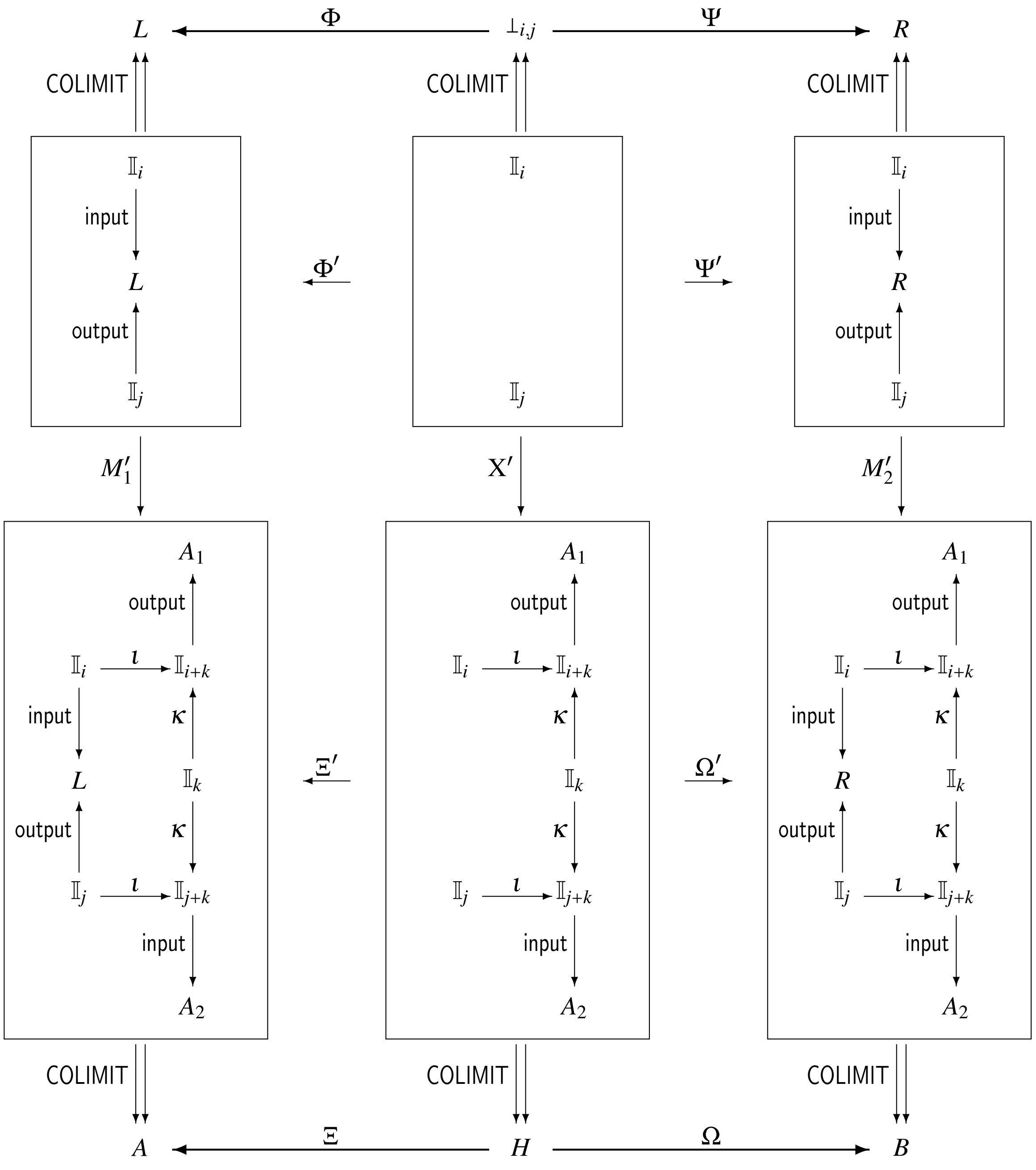
By ensuring that there is no "side entrance" from within the application graph A into the image of the LHS L, the dangling condition is crucial for the following result:

Lemma 4.3. Assume a solid term graph $L : \mathsf{TG}_{i,j}$ to be given, and let $\Phi : \perp_{i,j} \to L$ be the (necessarily-injective) DHG homomorphism from $\perp_{i,j}$ to L. If $A : \mathsf{TG}_{m,n}$ is a term graph and $M_1 : L \to A$ is an injective term graph matching that together with Φ satisfies the dangling condition, then there is an image context (k,A_1,A_2) for M_1. □

Such a context can be calculated in several different ways from the reachability in A, for example by collecting all edges into A_1 that are reachable from the input nodes of A via paths that do not touch the image of L under M_1. The difference $(A - A_1) - L$ would then induce A_2.

Sequential and parallel composition in the gs-monoidal category of term graphs (as morphisms) can be obtained as colimits in the category of DHG matchings. In the following diagram we denote the coproduct injections as ι and κ; for a $X : \mathsf{DHG}_{m,n}$ we use $\mathsf{input} : \mathbb{I}_m \to X$ as the DHG matching mapping $\mathbb{I}_m$ identically to the input nodes of X, and analogously $\mathsf{output} : \mathbb{I}_n \to X$.

The lower-left box below contains the diagram that has as its colimit the application graph A, factored into the context (k, A_1, A_2) and an image of the left-hand side L as $A \cong A_1 \mathbin{;} (L \otimes \mathbb{I}_k) \mathbin{;} A_2$.



The key observation is now that for a redex with $\bot$ as gluing graph and injective rule LHS Φ and injective matching M_1 satisfying the gluing condition, the DPO derivation step in the category of DHG matchings can be factored over a completely standard DPO diagram in a category of diagrams over the category of DHG matchings, as indicated in the nested diagram above.

The double-square diagram in the middle there **is a double pushout** in the category of diagrams over the category of DHG matchings with *rigid diagram homomorphisms*, which we define to be diagram homomorphisms that have only identity morphisms as components, or, in other words, that are node- and edge-label preserving graph homomorphisms between the underlying node- and edge-labelled graphs of the diagrams.

A key ingredient for this factoring to work is the restriction of the gluing graph to a "pure interface" $\perp_{i,j}$, so that it does not need to occur "in the place of L". It is crucial that this place is empty in the gluing and host diagrams, since otherwise we would not have rigid diagram homomorphisms horizontally.

As a result, since **the $\mathsf{COLIMIT}$ functor preserves pushouts**, the context decomposition carries over to the result B of the original DPO rewrite step, and we have:

$$B \quad \cong \quad A_1 \mathbin{⨟} (R \otimes \mathbb{I}_k) \mathbin{⨟} A_2$$

All this together proves:

Theorem 4.4. Let a DHG homomorphism span $L \xleftarrow{\Phi} \perp_{i,j} \xrightarrow{\Psi} R$ be given where L and R are term graphs. If $A : \mathsf{TG}_{m,n}$ is a term graph, $M_1 : L \to A$ is an injective term graph matching that together with Φ satisfies the dangling condition, and (k, A_1, A_2) is an image context for M_1, then the result graph B of the induced DPO in the category of DHG matchings is isomorphic to $A_1 \mathbin{⨟} (R \otimes \mathbb{I}_k) \mathbin{⨟} A_2$, that is, the same (k, A_1, A_2) is also an image context for the morphism $M_2 : R \to B$ resulting from the DPO. □

Note that this result is independent of the choice of image context for M_1. (Unlike for Theorem 3.3, we did not need to restrict Φ to be injective here. Injectivity of M_1 however is needed for the "image context for" statements according to Def. 4.2, and ultimately for making M_1' a rigid diagram homomorphism.)

Let us now assume a semantics to be chosen, that is, some gs-monoidal category (e.g., *Set*), and one of its objects $\mathcal{V}$ as interpretation of 1. We will use "$;$" as sequential composition and "$\times$" as parallel (that is, monoidal) composition in the semantics category.

For a term graph $J : \mathsf{TG}_{m,n}$, we write $[\![J]\!]_{m,n}$ for its semantics, which is a morphism from $\mathcal{V}^m$ to $\mathcal{V}^n$. In other words, we denote the morphism component of the semantics functor with $[\![\_]\!]$; since this is a gs-monoidal functor, we have in particular $[\![J_1 \mathbin{⨟} J_2]\!] = [\![J_1]\!] \mathbin{;} [\![J_2]\!]$ and $[\![J_1 \otimes J_2]\!] = [\![J_1]\!] \times [\![J_2]\!]$.

Under the assumption that the rule $L \longleftarrow \perp_{i,j} \longrightarrow R$ is semantics preserving, that is, $[\![L]\!]_{i,j} = [\![R]\!]_{i,j}$, we therefore easily obtain semantics preservation of the rewrite result:

$$\begin{aligned}
[\![A]\!]_{m,n} &= [\![A_1 \mathbin{⨟} (L \otimes \mathbb{I}_k) \mathbin{⨟} A_2]\!]_{m,n} \\
&= [\![A_1]\!]_{m,i+k} \mathbin{;} ([\![L]\!]_{i,j} \times [\![\mathbb{I}_k]\!]_{k,k}) \mathbin{;} [\![A_2]\!]_{j+k,n} \\
&= [\![A_1]\!]_{m,i+k} \mathbin{;} ([\![R]\!]_{i,j} \times [\![\mathbb{I}_k]\!]_{k,k}) \mathbin{;} [\![A_2]\!]_{j+k,n} \\
&= [\![A_1 \mathbin{⨟} (R \otimes \mathbb{I}_k) \mathbin{⨟} A_2]\!]_{m,n} \\
&= [\![B]\!]_{m,n}
\end{aligned}$$

For rules with $\perp_{i,j}$ as gluing graph, this, together with Theorem 4.4, allows us to extend Theorem 3.3 with semantics preservation:

Theorem 4.5. If a term graph rewrite rule formulated as a span $L \xleftarrow{\Phi} \bot_{i,j} \xrightarrow{\Psi} R$ of DHG homomorphisms with term graphs L, $R : \mathsf{TG}_{i,j}$, and with injective Φ, is applied via an injective term graph matching M_1 to an application term graph $A : \mathsf{TG}_{m,n}$, where M_1 together with Φ satisfies the dangling condition, then the diagram

$$\begin{tikzcd}
L \arrow[d, "M_1"'] & \bot_{i,j} \arrow[l, "\Phi"'] \arrow[r, "\Psi"] & R \\
A & &
\end{tikzcd}$$

can be completed to a TG-DPO

$$\begin{tikzcd}
L \arrow[d, "M_1"'] & \bot_{i,j} \arrow[l, "\Phi"'] \arrow[r, "\Psi"] \arrow[d, "X"'] & R \arrow[d, "M_2"] \\
A & H \arrow[l, "\Xi"'] \arrow[r, "\Omega"] & B
\end{tikzcd}$$

and for any gs-monoidal semantics functor $[\![\_]\!]$ for which the rule is semantics-preserving, that is, $[\![L]\!]_{i,j} = [\![R]\!]_{i,j}$, the resulting TG-DPO rewrite is also semantics-preserving, that is, $[\![A]\!]_{m,n} = [\![B]\!]_{m,n}$. □

5 Conclusion and Outlook

By considering a straight-forward adaptation of the DPO approach to term graph rewriting, we obtained an easily-understandable concept of rule application. By lifting this adapted DPO into a standard DPO of diagrams, we have been able to transfer the context decomposition from the left-hand side to the right-hand side, obviating the need to consider any semantics for general DHGs such as $\bot_{i,j}$. As result, we obtained a semantics preservation theorem that will be an important tool in the generation of verified code optimisation tools employing rule-based transformation of data-flow graphs, as outlined for example in [Kah14].

We originally started in [Kah11] to formalise term graphs essentially as defined in Sect. 2 in the dependently-typed programming language and proof assistant Agda [Nor07]. The current status of this project [Kah17, Zha18] includes term graph decomposition and a proof for its correctness, which essentially constitutes a machine-checked proof of the result of Corradini and Gadducci [CG99a] that term graphs form a free gs-monoidal category. As next steps, we plan to extend this development to cover also the results of the current paper, that is, definedness and semantics preservation of TG-DPO rewriting steps, and then to use this as a verified implementation of semantics-preserving term graph rewriting.

References

[AK09] C.K. Anand & W. Kahl (2009): *An Optimized Cell BE Special Function Library Generated by Coconut*. *IEEE Transactions on Computers* 58(8), pp. 1126–1138, doi:10.1109/TC.2008.223.

[AKP00] Z.M. Ariola, J.W. Klop & D. Plump (2000): *Bisimilarity in Term Graph Rewriting*. *Information and Computation* 156(1), pp. 2–24, doi:10.1006/inco.1999.2824.

[Ban93] R. Banach (1993): *A Fibration Semantics for Extended Term Graph Rewriting*. In Sleep et al., editors: *Term Graph Rewriting: Theory and Practice* [SPE93], chapter 7, pp. 91–100.

[CG97] A. Corradini & F. Gadducci (1997): *A 2-categorical presentation of term graph rewriting*. In E. Moggi & G. Rosolini, editors: *Category Theory and Computer Science*, *LNCS* 1290, Springer, Berlin, Heidelberg, pp. 87–105, doi:10.1007/BFb0026983.

[CG99a] A. Corradini & F. Gadducci (1999): *An Algebraic Presentation of Term Graphs, via GS-Monoidal Categories*. *Applied Categorical Structures* 7(4), pp. 299–331, doi:10.1023/A:1008647417502.

[CG99b] A. Corradini & F. Gadducci (1999): *Rewriting on cyclic structures: Equivalence between the operational and the categorical description*. *RAIRO Theor. Inform. Appl.* 33, pp. 467–493, doi:10.1051/ita:1999128.

[CG02] A. Corradini & F. Gadducci (2002): *Categorical rewriting of term-like structures*. *ENTCS* 51, pp. 108–121, doi:10.1016/S1571-0661(04)80195-6. GETGRATS Closing Workshop.

[CG05] A. Corradini & F. Gadducci (2005): *On Term Graphs as an Adhesive Category*. *ENTCS* 127(5), pp. 43–56, doi:10.1016/j.entcs.2005.02.014.

[CR93] A. Corradini & F. Rossi (1993): *Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming*. *Theoret. Comput. Sci.* 109(1–2), pp. 7–48, doi:10.1016/0304-3975(93)90063-Y.

[DEP09] D. Duval, R. Echahed & F. Prost (2009): *A Heterogeneous Pushout Approach to Term-Graph Transformation*. In R. Treinen, editor: *Rewriting Techniques and Applications*, Springer, Berlin, Heidelberg, pp. 194–208, doi:10.1007/978-3-642-02348-4_14.

[HP91] B. Hoffmann & D. Plump (1991): *Implementing Term Rewriting by Jungle Evaluation*. *Informatique théorique et applications/Theoretical Informatics and Applications* 25(5), pp. 445–472, doi:10.1051/ita/1991250504451.

[HP12] A. Habel & D. Plump (2012): *$\mathcal{M}$,$\mathcal{N}$-Adhesive Transformation Systems*. In H. Ehrig et al., editors: *Graph Transformation, ICGT 2012*, *LNCS* 7562, Springer, pp. 218–233, doi:10.1007/978-3-642-33654-6_15.

[KAC06] W. Kahl, C.K. Anand & J. Carette (2006): *Control-Flow Semantics for Assembly-Level Data-Flow Graphs*. In W. McCaull, M. Winter & I. Düntsch, editors: *8th Intl. Seminar on Relational Methods in Computer Science, RelMiCS 8, Feb. 2005*, *LNCS* 3929, Springer, pp. 147–160, doi:10.1007/11734673_12.

[Kah96] W. Kahl (1996): *Algebraische Termgraphersetzung mit gebundenen Variablen*. Reihe Informatik, Herbert Utz Verlag Wissenschaft, München. ISBN 3-931327-60-4; also Doctoral Diss. at Univ. der Bundeswehr München, Fakultät für Informatik.

[Kah97] W. Kahl (1997): *A Fibred Approach to Rewriting — How the Duality between Adding and Deleting Cooperates with the Difference between Matching and Rewriting*. Technical Report 9702, Fakultät für Informatik, Universität der Bundeswehr München. Available at `http://www.cas.mcmaster.ca/~kahl/Publications/TR/Kahl-1997b.html`.

[Kah11] W. Kahl (2011): *Dependently-Typed Formalisation of Typed Term Graphs*. In R. Echahed, editor: *Proc. of 6th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2011*, *EPTCS* 48, pp. 38–53, doi:10.4204/EPTCS.48.6.

[Kah14] W. Kahl (2014): *Towards "Mouldable Code" via Nested Code Graph Transformation*. *J. Logic and Algebraic Programming* 83(2), pp. 225–234, doi:10.1016/j.jlap.2014.02.010.

[Kah17] W. Kahl (2017): *Relation-Algebraic Theories in Agda — RATH-Agda-2.2*. Mechanically checked Agda theories, with 580 pages literate document output. http://relmics.mcmaster.ca/RATH-Agda/. With contributions by Musa Al-hassy and Yuhang Zhao.

[KKSV93] J.R. Kennaway, J.W. Klop, M.R. Sleep & F.J. de Vries (1993): *The Adequacy of Term Graph Rewriting for Simulating Term Rewriting*. In Sleep et al., editors: *Term Graph Rewriting: Theory and Practice* [SPE93], chapter 12, pp. 157–170.

[KKSV94] J.R. Kennaway, J.W. Klop, M.R. Sleep & F.J. de Vries (1994): *On the Adequacy of Graph Rewriting for Simulating Term Rewriting*. ACM Transactions on Programming Languages and Systems 16(3), pp. 493–523, doi:10.1145/177492.177577.

[Law63] F.W. Lawvere (1963): *Functorial Semantics of Algebraic Theories*. Proc. Nat. Acad. Sci. USA 50, pp. 869–872, doi:10.2307/2272673.

[ML71] S. Mac Lane (1971): *Categories for the Working Mathematician*. Springer-Verlag, doi:10.1007/978-1-4757-4721-8.

[Nor07] U. Norell (2007): *Towards a Practical Programming Language Based on Dependent Type Theory*. Ph.D. thesis, Dept. Comp. Sci. and Eng., Chalmers Univ. of Technology. See also http://wiki.portal.chalmers.se/agda/pmwiki.php.

[Plu02] D. Plump (2002): *Essentials of Term Graph Rewriting*. ENTCS 51, pp. 277–289, doi:10.1016/S1571-0661(04)80210-X. GETGRATS Closing Workshop.

[Ros93] K.H. Rose (1993): *Graph-based Operational Semantics of a Lazy Functional Language*. In Sleep et al., editors: Term Graph Rewriting: Theory and Practice [SPE93], chapter 22, pp. 303–316.

[SPE93] M.R. Sleep, M.J. Plasmeijer & M.C.J.D. van Eekelen, editors (1993): *Term Graph Rewriting: Theory and Practice*. Wiley.

[Wad71] C.P. Wadsworth (1971): *Semantics and Pragmatics of the Lambda Calculus*. D.Phil. thesis, Oxford University.

[Zha18] Y. Zhao (2018): *Formalisation of Term Graph Rewriting in Agda — TGR1*. Mechanically checked Agda development, with 283 pages literate document output. http://relmics.mcmaster.ca/RATH-Agda/TGR1/.