Relational Matching for Graphical Calculi of Relations

WOLFRAM KAHL

Department of Computing Science, German Armed Forces University Munich. 85577 Neubiberg, Germany

ABSTRACT

In this paper we extend an earlier approach to graphical relation calculi towards relational matching, thus allowing proofs with fewer auxiliary steps and concentrating more on the essential proof ideas.

For facilitating the formal argument we introduce *hierarchical relational diagrams* as an intermediate structure and employ more of the algebraic graph rewriting repertoire for defining relational rewriting of these hierarchical diagrams.

1. Introduction

Relational formalisations can be very concise and precise and can allow short, calculational proofs under certain circumstances. Examples can be found in [16, 18, 2].

In situations corresponding to the simultaneous use of many variables in predicate logic, however, either a style using predicate logic at the meta-level with point variables has to be adopted or impractical and clumsy manipulations of tuples have to be employed inside relation calculus. In the application of relational formalisation to term graphs with bound variables [8, 9, 12] we have been forced to employ both methods extensively, and, independently of other approaches [3, 4], have been driven to develop a *graphical calculus* for making complex relation algebraic proofs more accessible.

In [10] we presented this graphical calculus as a formalism for relation algebraic diagrams (graphs) with a semantics that allows interpretation of these diagrams as relation algebraic formulae. We defined a rewriting mechanism for relational diagrams and showed how rewriting sequences could be used as proofs. The matching concept of the rewriting mechanism consists of — more or less standard — functional graph homomorphisms. The rewriting mechanism itself is a variant of the algebraic approach to graph rewriting (see [5] for a tutorial overview). This background makes our approach more general and more flexible than the other approaches presented in the literature [3, 4], although otherwise it does of course share many common points with them.

In this paper we extend the approach of [10] to hierarchical diagrams and relational matching, thus allowing proofs with fewer auxiliary steps. These proofs therefore concentrate more on the essential proof ideas and continue the trend of [10] of avoiding technical transformations in the proofs that only serve to prepare more essential proof steps.

We start this paper, after some further motivation in Sect. 2, with a recapitulation of the essential definitions from [10] in Sect. 3. We then proceed to take advantage of the generality of these definitions for achieving an comparatively easy transition to relational matching, presented in Sect. 4.

2. Motivation

At least for simple relational expressions that employ only intersection and composition of relations, it is easy to come up with a graphical representation; for example the two sides of the inclusion

$$(A;\Theta \sqcap B);\Theta \sqsubseteq (A \sqcap B;\Theta);\Theta \ ,$$

which holds whenever Θ is an equivalence relation, can be depicted as the leftmost and rightmost graphs in the following picture (with orientation from the lower left to the upper right):

$$\begin{tikzcd}
\beta \arrow[r, "\Theta"] & \beta \arrow[r, "\Theta"] & \beta & & \beta \arrow[r, "\Theta"] & \beta \arrow[r, "\Theta"] \arrow[d, "\Theta"] & \beta & & & \beta \arrow[d, "\Theta"] & \beta \\
\alpha \arrow[u, "A"] \arrow[ur, "B"'] & & & \longrightarrow & \alpha \arrow[u, "A"] \arrow[ur, "B"] \arrow[r, "A"] & \beta \arrow[ur, "\Theta"] & & \leftarrow\cdots & \alpha \arrow[ur, "B"] \arrow[r, "A"] & \beta \arrow[ur, "\Theta"] &
\end{tikzcd}$$

Here α and β are type variables, and A and B are relation variables. At least for diagrams as simple as these two, it is also obvious which relational expressions they correspond to.

When applying such an inclusion rule $L \sqsubseteq R$ in a relational proof, one is never sure whether it is safe to throw L away or whether one should keep it around, using the inclusion $L \sqsubseteq L \sqcap R$, which then is in fact an equality.

Since it is always possible to throw superfluous intersection terms away as long as one is only interested in an inclusion, a safe policy is to keep the left-hand sides around until it is clear that they are not needed anymore.

Therefore we chose to let the graph rule consist of the left and middle graphs in the picture above together with the embedding between them.[1]

Matching of a rule into an application diagram obviously should just be another diagram morphism, so the natural choice of rewrite mechanism is the single pushout in the category of total diagram morphisms.[2]

With the example rule above, we can obtain the following rewrite step:

$$\begin{tikzcd}
 & \beta \arrow[r, "\Theta"] & \beta \arrow[r, "\Theta"] & \beta & {} \arrow[r, "r"] & {} & & \beta \arrow[r, "\Theta"] \arrow[d, phantom] & \beta \arrow[r, "\Theta"] \arrow[d, "\Theta"] & \beta \\
 & \alpha \arrow[u, "A"] \arrow[ur, "B"'] & & & & & & \alpha \arrow[u, "A"] \arrow[ur, "B"'] \arrow[r, "A"] & \beta \arrow[ur, "\Theta"'] & \\
 & & {} \arrow[d, "f"'] & & & & & & {} \arrow[d, "g"'] & \\
 & & {} & & & & & & {} & \\
\gamma \arrow[d, "Y"'] & \beta \arrow[l, "K"'] \arrow[r, "\Theta"] \arrow[dl, "H"'] & \beta \arrow[r, "\Theta"] & \beta \arrow[d, "Z^*"'] & {} \arrow[r, "s"] & {} & \gamma \arrow[d, "Y"'] & \beta \arrow[l, "K"'] \arrow[r, "\Theta"] \arrow[dl, "H"'] & \beta \arrow[r, "\Theta"] \arrow[d, "\Theta"] & \beta \arrow[d, "Z^*"] \\
\delta & \alpha \arrow[u, "S^{\smile}"] \arrow[ur, "P"'] & & \beta & & & \delta & \alpha \arrow[u, "S^{\smile}"] \arrow[ur, "P"'] \arrow[r, "S^{\smile}"'] & \beta \arrow[ur, "\Theta"'] & \beta
\end{tikzcd}$$

2.1. Example Derivation

To show the usefulness of this approach, let us resort to an example from an application, a part of the proof of [8, Lemma 4.2.3]. Let us assume an object O and relations C, V and W in the underlying relation algebra $\mathcal{R}$ for which the following two rules are correct:

[1] Actually, the diagram rule given here is a little bit stronger than the original inclusion, in that we did not draw an independent second "B"-edge, but this stronger version follows easily from the symmetry of the equivalence relation Θ and is easier to draw.

[2] In a category, for three objects A, B, and C and two arrows $f{:}A \to B$ and $g{:}A \to C$, a *pushout* is an object D together with two arrows $h{:}B \to D$ and $j{:}C \to D$ with $f;h = g;j$, such that for every object D' together with two arrows $h'{:}B \to D'$ and $j'{:}C \to D'$ with $f;h' = g;j'$ there is a unique arrow $u{:}D \to D'$ such that $h;u = h'$ and $j;u = j'$. For an introduction to the use of the pushout concept in graph rewriting see [5].

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"] & B \arrow[d, "h"] \\
C \arrow[r, "j"] & D
\end{tikzcd}$$

$W ; C^{\smile} \sqsubseteq C ; W^{\smile}$

(i)

$$\begin{tikzcd}
O & \\
O \arrow[u, "C"] & O \arrow[ul, "W"'] \\
& O \arrow[ul, "W"] \arrow[u, "C"']
\end{tikzcd}$$

$C ; V^{\smile} \sqsubseteq V^{\smile} ; C$

(ii)

$$\begin{tikzcd}
O \arrow[r, "V"] & O \\
O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[u, "C"']
\end{tikzcd}$$

For both rules we have explicitely drawn only the right hand side; the left hand sides are the subgraphs induced by the boldened edges — as long as the rule morphism is injective, this abbreviating method of representation is possible. (The different layout of the two rules has been chosen for better fitting into their application below. Furthermore note that the first rule could also have been read $C ; W^{\smile} \sqsubseteq W^{\smile} ; C$ — the rules are valid no matter which orientation of the left-hand side is considered.)

Now, for a derivation of $\mathbb{T} ;(C \sqcap V ; W) \sqsubseteq \mathbb{T} ;(C \sqcap V ; W) ; C$, first these two rules are applied in order and then we restrict the result to a subgraph that is sufficient for our purposes:

$$\begin{tikzcd}
\mathbb{1} \arrow[r, dotted, "\mathcal{I}"] & O & \\
& O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[ul, "W"']
\end{tikzcd}
\quad
\begin{tikzcd}
\mathbb{1} \arrow[r, dotted, "\mathcal{I}"] & O & \\
& O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[ul, "W"'] \\
& & O \arrow[ul, "W"] \arrow[u, "C"']
\end{tikzcd}
\quad
\begin{tikzcd}
\mathbb{1} \arrow[r, dotted, "\mathcal{I}"] & O & \\
& O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[ul, "W"'] \\
& O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[ul, "W"'] \arrow[u, "C"']
\end{tikzcd}
\quad
\begin{tikzcd}
\mathbb{1} \arrow[r, dotted, "\mathcal{I}"] & O & \\
& O \arrow[u, "C"] & \\
& O \arrow[u, "C"] \arrow[r, "V"] & O \arrow[ul, "W"']
\end{tikzcd}$$

A standard inclusion chain for the same proof takes considerably more steps:

$$\begin{array}{lll}
 & \mathbb{T} ;(C \sqcap V ; W) & \text{(1)} \\
\sqsubseteq & \mathbb{T} ; V ;(W \sqcap V^{\smile} ; C) & \text{modal rule} \\
\sqsubseteq & \mathbb{T} ;(V^{\smile} \sqcap W ; C^{\smile}) ; C & \mathbb{T} ; V \sqsubseteq \mathbb{T}\text{, modal rule} \\
\sqsubseteq & \mathbb{T} ;(V^{\smile} \sqcap C ; W^{\smile}) ; C & \text{(2): (i)} \\
\sqsubseteq & \mathbb{T} ; C^{\smile} ;(W \sqcap C ; V^{\smile}) ; C & \text{modal rule} \\
\sqsubseteq & \mathbb{T} ;(W \sqcap V^{\smile} ; C) ; C & \mathbb{T} ; C^{\smile} \sqsubseteq \mathbb{T}\text{, (3): (ii)} \\
\sqsubseteq & \mathbb{T} ; V^{\smile} ;(C \sqcap V ; W) ; C & \text{modal rule} \\
\sqsubseteq & \mathbb{T} ;(C \sqcap V ; W) ; C & \mathbb{T} ; V^{\smile} \sqsubseteq \mathbb{T}
\end{array}$$

$$\begin{tikzcd}
\mathbb{1} \arrow[r, dotted, "\mathcal{I}"] & O & \\
& O \arrow[u, "C_1"] \arrow[r, "V_1"] & O \arrow[ul, "W_1"'] \\
& O \arrow[u, "C_3"] \arrow[r, "V_3"] & O \arrow[ul, "W_2"'] \arrow[u, "C_2"']
\end{tikzcd}$$

(On the right we have shown a different way to present the diagrammatic proof: we draw just one graph and additionally annotate the edges with their “generation”; we also boldened the edges needed in the result.)

Obviously, the diagram proof is simpler and more intuitive than the linear (term) proof. The main reason for this are the frequent “changes of point of view” that are reflected most typically in applications of modal rules. These “changes of point of view” together with references to many previously introduced nodes are the main effects that make the kind of diagram proofs as seen so far (i.e., in the manner of [10]) shorter and easier to read than linear proofs.

2.2. Extension to Relational Matching

Within the framework presented in [10], however, the following desirable rewriting step is impossible:

$$\begin{tikzcd}
& \beta \arrow[r,"\Theta"] & \beta \arrow[r,"\Theta"] & \beta & & & \beta \arrow[r,"\Theta"] & \beta \arrow[r,"\Theta"] \arrow[dr,"\Theta"] & \beta \\
& & \alpha \arrow[ul,"A"] \arrow[u,"B"'] & & {} \arrow[r] & {} & & \alpha \arrow[ul,"A"] \arrow[u,"B"'] \arrow[r,"A"] & \beta \arrow[u,"\Theta"'] \\
& & {} \arrow[d] & & & & & {} \arrow[d] & \\
& & & \beta & & & & & \beta \\
& \beta \arrow[r,"\Theta"] & \beta \arrow[ur,"\Theta"] & & {} \arrow[r] & {} & \beta \arrow[r,"\Theta"] & \beta \arrow[r,"\Theta"] \arrow[ur,"\Theta"] & \beta \arrow[u,"\Theta"'] \\
\gamma \arrow[ur,"S"] \arrow[r,"R"] & \delta \arrow[u,"D"'] & \alpha \arrow[l,"C"'] \arrow[u,"B"'] & & & \gamma \arrow[ur,"S"] \arrow[r,"R"] & \delta \arrow[u,"D"'] & \alpha \arrow[l,"C"'] \arrow[u,"B"'] \arrow[r,"C"] & \delta \arrow[u,"D"']
\end{tikzcd}$$

Here the edge labelled with the relation variable A is itself considered to be a variable, and it is matched so-to-speak to the whole gray sub-diagram consisting of the two edges labelled C and D.

For emulating such a rewriting step, in the absence of the R-edge the following intermediate steps would be sufficient:

- Contraction of the sub-diagram into one $C ; D$ edge
- Rule application
- Expansion of now two $C ; D$ edges

In the presence of the R-edge, however, even more care has to be taken: the sub-diagram would not be contracted but instead an additional $C ; D$ edge created, which would have to be removed again after rule application — in more complex situations quite some bookkeeping would be involved in order to properly undo all temporary modifications. Obviously an approach allowing to do all this in one step would be much more satisfying.

A possible direct solution to this problem might be to define relational matchings with several edges in the image of one edge, and then to find an appropriate category where composition of such relational matchings is well-defined and well-behaving; finally several correctness proofs would have to be redone for being able to substitute relational matchings for functional matchings in the approach of [10].

This direct solution looks rather nontrivial, and a concept of relational matching where one pattern edge is related to several (contiguous) edges in the application graph while the nodes in-between are not related to anything is not very satisfactory.

Instead we shall adopt an *indirect solution*: We shall define *hierarchical graphs* with whole sub-diagrams inside edge labels of other diagrams — such sub-diagrams can be *flattened* via double-pushout rewriting steps. Since matchings are only used in the vertical arrows, we can employ a heterogeneous rewriting square (related to those of [1, 11]), and relational matchings then do not need to give rise to a category, i.e., we need not make sure that composition of two relational matchings is again a relational matching. For rewriting we shall introduce a stacked single pushout rewriting step, so on the whole our approach is very modular, fitting together small pieces, that are themselves not too complicated, into a very powerful diagram rewriting concept.

Before we get to do this, however, we have to review the basics of [10].

3. Basics

In this section we recapitulate the basic definitions and results of [10]. We do this in considerable detail and also provide key motivations for facilitating the access to the remainder of the paper.

First we recapitulate the relation algebraic notation in 3.1, then we start with simple terms in 3.2. These are then brought into the labels of relational diagrams in 3.3, where the concepts of rules and rewriting are introduced, too. In 3.4, finally, we deal with the correctness of our graphical calculus.

3.1. Notation

The structure we exploit in our diagram proofs is that of a locally complete unitary pretabular allegory (LCUPA) [7], which is essentially an abstract relation algebra in the sense of [16] (without negation), equipped with all relational products (which are the direct products in the underlying category of total functions).

For improving understandability we shall use the more widely accepted nomenclature of abstract relation algebra (rather than that of LCUPAs, which is employed by [3]) and also its notation as agreed upon in [2]. A relation algebra has at its core a (self-dual) category with objects $A, B, \ldots$ and morphisms $R, S, \ldots$; we call the morphisms **relations**. **Composition** of two relations $R : A \leftrightarrow B$ and $S : B \leftrightarrow C$ is written $R;S$; the **converse** of a relation $R : A \leftrightarrow B$ is $R^{\smile} : B \leftrightarrow A$; the

complement (or **negation**) of a relation $R : A \leftrightarrow B$ is $\overline{R} : A \leftrightarrow B$; **intersection** of two relations $R, S : A \leftrightarrow B$ is $R \sqcap S$, and their **union** is $R \sqcup B$; for any object A, the **identity relation** is $\mathbb{I}_A$; for two objects A and B the **universal relation** is $\mathbb{T}_{A,B}$, and the **empty relation** is $\mathbb{L}_{A,B}$. Inclusion of $R : A \leftrightarrow B$ in $S : A \leftrightarrow B$, i.e. the fact that $R \sqcap S = R$, is denoted by $R \sqsubseteq S$.

Among the binary operators, relational composition "$;$" has higher priority than union "$\sqcup$" and intersection "$\sqcap$".

For the **relational product** $A \times B$ of two objects A and B, $\pi_{A\times B}$ and $\rho_{A\times B}$ denote the first resp. second **projection mapping**. For two products $A \times B$ and $C \times D$ and two relations $R : A \leftrightarrow C$ and $S : B \leftrightarrow D$, the product $(R||S) : (A\times B) \leftrightarrow (C\times D)$ of R and S is defined as $(R||S) = \pi_{A\times B} ; R ; \pi^{\smile}_{C\times D} \sqcap \rho_{A\times B} ; S ; \rho^{\smile}_{C\times D}$.

The laws that are required to hold are the usual laws of relation calculus which we do not restate here.

For a set A we denote the set of **finite sequences** of elements of A with A^*. For a function $f : X \to Y$, we denote the mapping of f to sequences by $f^* : X^* \to Y^*$.

A sequence $\langle A_1, \ldots, A_n\rangle$ of objects that is itself considered as an object is understood to be the corresponding finite product $(A_1 \times \cdots \times A_n)$.

We write set-comprehensions according to the Z-notation [17], which uses the pattern "{ *signature* | *predicate* • *term* }" instead of the otherwise frequently observed pattern "{ *term* | *predicate* }". So we have, for example, $\{n : \mathbb{N} \mid n < 4 \bullet n^2\} = \{0, 1, 4, 9\}$.

3.2. *Type and Relation Terms*

We first introduce type terms and relation terms as the syntactic basis of our calculus. We reuse the operator symbols introduced above, but we shall employ "$\equiv$" for syntactical equality of terms.

Definition 3.1 A **type term** can be a *type constant*, including $\mathbb{1}$ for the unit type, a *type variable* ($\alpha, \beta, \gamma, \ldots$), a *product type* $T \times U$ of two type terms T and U, or a *constructor type* $C(T_1, \ldots, T_n)$ created from n type terms $T_1, \ldots, T_n$ by application of an n-ary *type term constructor* C. □

Obviously, the product type could be considered as just another constructor type, but since it has a special status here, we rather treat it separately.

A **type substitution** is a finite domain partial function from type variables to type terms. Application of a type substitution is defined as usual.

Definition 3.2 A **relation term** of type $A \leftrightarrow B$ for two type terms A and B can be

- a *relation constant*, including $\mathbb{I}$ (if $A \equiv B$), $\mathbb{T}$, $\mathbb{L}$, π (if there is a type term C such that $A \equiv B \times C$), and ρ (if there is a type term C such that $A \equiv C \times B$),

- a *relation variable*,
- the *converse* $R^{\smile}$ of a relation term R of type $B \leftrightarrow A$ (written $R : B \leftrightarrow A$),
- the *negation* $\overline{R}$ of a relation term $R : A \leftrightarrow B$,
- the *composition* $R;S$ of two relation terms $R : A \leftrightarrow C$ and $S : C \leftrightarrow B$,
- the *intersection* $R \sqcap S$ or the *union* $R \sqcup S$ of two relation terms $R : A \leftrightarrow B$ and $S : A \leftrightarrow B$,
- a *constructor term* $c(R_1, \ldots, R_n)$ created from n relation terms R_i by application of an n-ary *relation term constructor* c, with type constraints on the R_i depending on c.

Additionally, in any composite term, all occurrences of a relation variable must be of the same type. □

A **relation substitution** is a finite domain partial function from relation variables to relation terms. Application of a substitution is again defined as usual.

Finally, an atomic **relational formula** is either an equality $R = S$ or an inclusion $R \sqsubseteq S$ for two relational terms R and S of the same type.

3.3. Syntax of Relational Diagrams

We now introduce *relational diagrams* as a special kind of labelled graphs or hypergraphs. Although hypergraphs are of course more general, we include the graph case for offering the reader a smoother access:

Definition 3.3 A **relational diagram** is a labelled directed (hyper-)graph $(\mathcal{N}, \mathcal{E}, \mathbf{s}, \mathbf{t}, \mathbf{n}, \mathbf{e})$ with node set $\mathcal{N}$, edge set $\mathcal{E}$, source mapping $\mathbf{s} : \mathcal{E} \to \mathcal{N}$ (resp. $\mathbf{s} : \mathcal{E} \to \mathcal{N}^*$), target mapping $\mathbf{t} : \mathcal{E} \to \mathcal{N}$ (resp. $\mathbf{t} : \mathcal{E} \to \mathcal{N}^*$); furthermore $\mathbf{n}$ is the node labelling, assigning every node a type term, and $\mathbf{e}$ is the edge labelling, assigning every edge a relation term of type $\mathbf{n}(\mathbf{s}(e)) \leftrightarrow \mathbf{n}(\mathbf{t}(e))$ (resp. $\mathbf{n}^*(\mathbf{s}(e)) \leftrightarrow \mathbf{n}^*(\mathbf{t}(e))$).

All occurrences of a relation variable in the range of $\mathbf{e}$ must be of the same type. □

For any relational diagram $G = (\mathcal{N}, \mathcal{E}, \mathbf{s}, \mathbf{t}, \mathbf{n}, \mathbf{e})$ we define its **emptied diagram** as the corresponding discrete graph: $G^0 := (\mathcal{N}, \emptyset, \emptyset, \emptyset, \mathbf{n}, \emptyset)$.

Homomorphisms between relational diagrams are defined as homomorphisms of labelled graphs with the additional possibility of substitutions in the labels:

Definition 3.4 A **simple relational diagram homomorphism** f from one relational diagram G_1 to another G_2 is a pair $(f_{\mathbf{n}}, f_{\mathbf{e}})$ of functions, with

- $f_{\mathbf{n}} : \mathcal{N}_1 \to \mathcal{N}_2$, $f_{\mathbf{e}} : \mathcal{E}_1 \to \mathcal{E}_2$,
- $\mathbf{s}_2(f_{\mathbf{e}}(a)) = f_{\mathbf{n}}(\mathbf{s}_1(a))$, $\mathbf{t}_2(f_{\mathbf{e}}(a)) = f_{\mathbf{n}}(\mathbf{t}_1(a))$, or, in the case of hypergraphs, $\mathbf{s}_2(f_{\mathbf{e}}(a)) = f^*_{\mathbf{n}}(\mathbf{s}_1(a))$, $\mathbf{t}_2(f_{\mathbf{e}}(a)) = f^*_{\mathbf{n}}(\mathbf{t}_1(a))$,
- there is a type substitution τ such that $\mathbf{n}_2(f_{\mathbf{n}}(v)) \equiv \tau(\mathbf{n}_1(v))$ for all nodes v,
- there is a relation substitution σ with $\mathbf{e}_2(f_{\mathbf{e}}(a)) \equiv \sigma(\mathbf{e}_1(a))$ for all edges a.

f is called **plain** if τ and σ can be set to empty substitutions. □

For graph rules, we have an inclusion semantics "$L \sqsubseteq R$" in mind. Since it is usually advisable to preserve previously established information, we arrive at a useful rule concept with just a plain homomorphism between the rule sides:

Definition 3.5 A **rule** $(L \xrightarrow{r} R)$ consists of two relational diagrams L and R together with a plain homomorphism r from L to R. □

According to the definition of rules as single homomorphisms, rewriting is defined by a single pushout construction — the difference to the single-pushout approach of [13, 14] is that here we still consider *total* homomorphisms:

Definition 3.6 A **rewrite step** for a rule $(L \xrightarrow{r} R)$ and a relational diagram G together with a homomorphism f from L to G is the pushout

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "f"'] & R \arrow[d, "g"'] \\ G \arrow[r, "s"] & H \end{tikzcd}$$

of r and f; the **result diagram** is the pushout object H.

A **derivation** of $H := G_n$ from $G := G_0$ is a sequence of rewrite steps

$$\begin{tikzcd} L_i \arrow[r, "r_i"] \arrow[d, "f_i"'] & R_i \arrow[d, "g_i"'] \\ G_{i-1} \arrow[r, "s_i"] & G_i \end{tikzcd}$$

and we let the **derivation morphism** be $s_1 ; \ldots ; s_n$. □

3.4. Relative Correctness of the Graphical Calculus

When considering the meaning of a diagram, a "direction" has to be imposed from the outside, and we use *interfaces* for this purpose:

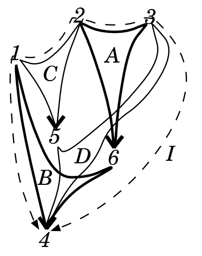
Definition 3.7 An **interface** (I, j) for a relational diagram G consists of a relational diagram I and a homomorphism j from I^0 to G, where I has no isolated nodes and exactly one edge ($|\mathcal{E}_I| = 1$), which has to be labelled with a variable.

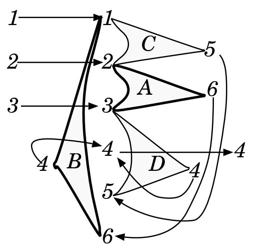
An interface (I, j) for G can be composed with a homomorphism k from G to H, yielding the interface $(I, j) ; k := (I, j ; k)$ for H. □

For keeping the whole approach as simple and as modular as possible, we only concern ourselves with correctness of the graphical calculus of relations relatively to the conventional calculus of relations. Therefore, the "semantics" of a relational diagram when seen through an interface is a relation term of appropriate type which appropriately assembles the relation terms to be found as the edge labels of the diagram in question.

Our aim is to construct such a relational expression in a canonic way — using the laws of relational calculus with products, it always may be transformed into equivalent expressions that may be more appealing for one or the other reason.

Below, for an example, to the left a relational diagram G consisting of four hyperedges is shown together with a three-input, one-output interface $\mathcal{I}$. To the right, we have drawn an intermediate diagram that will serve as guideline to the construction of the semantics of G under the interface $\mathcal{I}$, which we call the "readout" of G along $\mathcal{I}$ and which we denote $G_{\lceil\mathcal{I}\rceil}$. All simple edges there are labelled with the identity $\mathbb{I}$. Collapsing those edges returns the original diagram G, so the two should obviously considered as equivalent.





Assuming an appropriately nested input product and using the nested-pair isomorphism $\mathsf{PAass} : \alpha \times (\beta \times \gamma) \to (\alpha \times \beta) \times \gamma$, the readout obviously has to be equivalent to

$$(\mathbb{I} \| A) ; B \sqcap \mathsf{PAass} ; (C \| \mathbb{I}) ; D \ .$$

For abbreviating the formal treatment, we consider a sequence $\langle T_1, \ldots, T_n \rangle$ of type terms as denoting the product term $T_1 \times \cdots \times T_n$. For the image of a set $S : \mathbb{P}(A)$ under a function $f : A \to B$ we just write $f(S) : \mathbb{P}(B)$.

Definition 3.8 (Readout) Let a relational diagram G and an interface $\mathcal{I} = (I, j)$ for G with the one hyperedge $a_{\mathcal{I}}$ be given. Let $\langle m_1, \ldots, m_{\text{ex}} \rangle$ be a sequentialisation of the nodes of G outside the range of j. Then let $\mathcal{T}_{G,\mathcal{I}}$ be the following type term representing all nodes of the interface together with all other nodes of G:

$$\mathcal{T}_{G,\mathcal{I}} :\equiv \mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}}))) \times \mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}}))) \times \mathbf{n}^*(\langle m_1, \ldots, m_{\text{ex}} \rangle)$$

Furthermore, let $\pi_{\text{in}} : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}})))$ and $\pi_{\text{out}} : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}})))$ be the projections onto the source and target of the interface image. For every node $x \in \mathcal{N}_I$ of the interface diagram, let $\pi_x : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}(j(x))$ be the projection onto the component of x, and for every node sequence $s \in \mathcal{N}^*$, let $\pi_s : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}^*(s)$ be a projection onto the components of $\mathcal{T}_{G,\mathcal{I}}$ corresponding to s.

The **readout** of G via $\mathcal{I}$ is now a relation term $G_{\lceil\mathcal{I}\rceil}$ with the typing

$$G_{\lceil\mathcal{I}\rceil} : \mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}}))) \leftrightarrow \mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}}))) \ .$$

The readout is defined as:

$$\begin{aligned} G_{\lceil\mathcal{I}\rceil} :\equiv \pi_{\mathrm{in}}^{\smile};(\ & \textstyle\bigsqcap\{a : \mathcal{E} \bullet (\pi_{\mathbf{s}(a_{\mathcal{I}})};\mathbf{e}(a) \sqcap \pi_{\mathbf{t}(a_{\mathcal{I}})});\mathbb{T}\} \\ & \sqcap \textstyle\bigsqcap\{x, y : \mathcal{N}_I | x \neq y \wedge j(x) = j(y) \bullet (\pi_x \sqcap \pi_y);\mathbb{T}\} \\ & \sqcap \pi_{\mathrm{out}}) \end{aligned} \qquad \square$$

(See [10] for a discussion of the irrelevance of the choices involved.)

In the example above, the three main layers of nodes in the right diagram obviously correspond to the input type $\mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}})))$, the all-nodes type $\mathcal{T}_{G,\mathcal{I}}$ and the output type $\mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}})))$ respectively. The nodes ending the hyperedges can be regarded as the input of $\mathbb{T}$ in the first set component of $G_{\lceil\mathcal{I}\rceil}$; the readout is

$$\pi_{123}^{\smile};((\pi_{23};A\sqcap\pi_6);\mathbb{T}\sqcap(\pi_{16};B\sqcap\pi_4);\mathbb{T}\sqcap(\pi_{12};C\sqcap\pi_5);\mathbb{T}\sqcap(\pi_{53};D\sqcap\pi_3);\mathbb{T}\sqcap\pi_4)$$

This is equivalent to

$$\pi_{123}^{\smile};((\pi_1;\pi^{\smile}{}_1 \sqcap \pi_{23};A;\pi^{\smile}{}_6);\pi_{16};B \sqcap (\pi_{12};C;\pi^{\smile}{}_5 \sqcap \pi_3;\pi^{\smile}{}_3);\pi_{53};D);\pi_4$$

and again (modulo nesting of the input product) to the relation term initially proposed as readout of the example diagram. The second set component from the readout definition is empty here, since the interface is injective; otherwise there would be additional $\mathbb{I}$-edges between nodes of the middle layer.

We now start considering an arbitrary set $\mathcal{R}$ of additional axioms besides those for relation algebras with products, mainly for additional relation constants and relation term constructors. Since we have reserved the symbol "$\equiv$" for syntactic equality of terms, we can use "$=$" and "$\sqsubseteq$" freely for forming formulae. We shall write "$\mathcal{R} \vdash R = S$" resp. "$\mathcal{R} \vdash R \sqsubseteq S$" if the respective equality or inclusion can be derived using the axioms in $\mathcal{R}$ and those of relation algebras with products.

An important result about the readout construction is that plain homomorphisms can only decrease the semantics.

Theorem 3.9 $\mathcal{R} \vdash G_{\lceil\mathcal{I};k\rceil} \sqsubseteq K_{\lceil\mathcal{I}\rceil}$ holds for every interface $\mathcal{I}$ for K and every plain homomorphism k from K to G.

Accordingly, for every interface $\mathcal{I}$ for G and every subgraph K of G with natural injection k, whenever $j(I^0) \subseteq k(K)$ then $\mathcal{R} \vdash G_{\lceil\mathcal{I}\rceil} \sqsubseteq K_{\lceil\mathcal{I};k^{-1}\rceil}$. $\square$

Based on the readout, the appropriate concept of admissibility of rules is the following:

Definition 3.10 A rule $(L \xrightarrow{r} R)$ is **correct** if for all interfaces (I, j) for L,

$$\mathcal{R} \vdash L_{\lceil(I,j)\rceil} = R_{\lceil(I,j;r)\rceil} \quad . \qquad \square$$

It is not difficult to construct concrete rules where this equality holds for some interfaces, but not for others, and where application of these rules leads to invalid proofs. This equality is, however, guaranteed to hold for all interfaces for L whenever it holds for any interface (I, j) for L where j is surjective on the nodes.

The central result of [10] is that application of correct rules yields derivations which preserve the semantics of the readout of any interface into the starting diagram:

Theorem 3.11 Let an interface (I, j) for a relational diagram G and $(L \xrightarrow{r} R)$ with a matching homomorphism f from L to G be given, and consider the rewriting step yielding the pushout object H and the homomorphism s from G to H, then we have $\mathcal{R} \vdash G_{\lceil (I,j) \rceil} = H_{\lceil (I, j\,;\, s) \rceil}$.

Accordingly, for every interface (I, j) for the starting diagram G of a derivation from G to H with derivation morphism σ, we have $\mathcal{R} \vdash G_{\lceil (I,j) \rceil} = H_{\lceil (I, j\,;\, \sigma) \rceil}$. $\square$

With all this, the natural strategy for finding a proof of the inclusion formula $R \sqsubseteq S$ as a graph derivation on relational diagrams, as already applied in the example in 2.1, is the following:

i) Construct a diagram G with an interface (I, j) such that $\mathcal{R} \vdash R = G_{\lceil (I,j) \rceil}$.

ii) Suitably derive H from G with derivation morphism σ.

iii) Factorise σ into plain homomorphisms σ' from G to a suitable diagram H' and k from H' to H

iv) Recognise H' as a diagram with $\mathcal{R} \vdash S = H'_{\lceil (I,j') \rceil}$ where $j' = j\,;\,\sigma'$.

Only in rare cases the full derivation result H will be needed (yielding an equality); usually only an inclusion is required anyway.

4. Hierarchical Diagrams and Relational Matching

The idea of relational matching for our diagrams is that variable-labelled hyperedges in the rule can be matched not only to single hyperedges in the application diagram, but to whole sub-diagrams.

For being able to properly define relational matching and rewriting based on relational matching, we use the trick to employ hierarchical diagrams as an intermediate structure. For this purpose, the generality of our previous definitions comes in extremely handy, and more of the repertoire of algebraic graph rewriting can be employed in a very natural manner.

Definition 4.1 A **pointed diagram** of type $A \leftrightarrow B$ is a pair $(G, (I, j))$ consisting of a diagram G together with an interface (I, j) for G (with the one hyperedge $a_{\mathcal{I}}$), if $A \equiv \mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}})))$ and $B \equiv \mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}})))$. $\square$

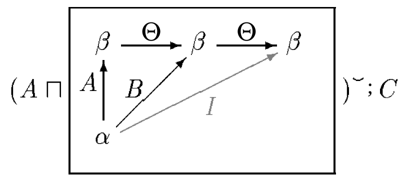
As an example we show the left-hand side of the rule from Sect. 2 as a pointed diagram:

$$\begin{tikzcd}
\beta \arrow[r, "\Theta"] & \beta \arrow[r, "\Theta"] & \beta & \bullet \arrow[l] \\
\alpha \arrow[u, "A"] \arrow[ur, "B"'] & \bullet \arrow[l] \arrow[urr, "I"'] & &
\end{tikzcd}$$

Here I consists of the whole graph induced by the edge labelled "I" in the drawing; the dotted arrows represent j.

From now on we consider the definition 3.2 of relation terms of type $A \leftrightarrow B$ to be extended to include pointed diagrams of type $A \leftrightarrow B$. We call a relation term that may contain pointed diagrams as some subterms an **extended relation term** while the original version will be called **simple relation term**.

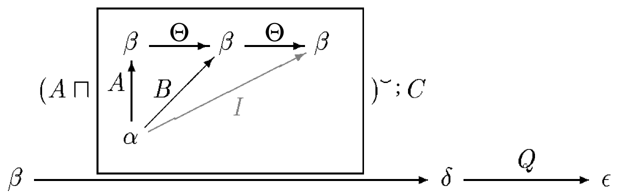
As an example extended relation term we insert the pointed diagram from above into some context:



Here we have drawn the interface edge directly into the diagram; this is more concise and — as long as j is injective — does not obscure any relevant information.

This gives of course rise to a mutual recursion between relation terms and the relational diagrams of Def. 3.3, which are now **hierarchical diagrams**.

In the next example, a pointed diagram occurs inside a relation term labelling an edge of the diagram under consideration:



The readout of Def. 3.8 now in general only yields an extended relation term; therefore we introduce another concept that maps any extended relation term to a corresponding simple relation term:

Definition 4.2 The **unfolding** of an extended relation term t, denoted $[\![t]\!]$, is defined as follows:

- if t is a relation constant or relation variable, then $[\![t]\!] :\equiv t$,
- $[\![t^{\smile}]\!] :\equiv [\![t]\!]^{\smile}$ and $[\![\overline{t}]\!] :\equiv \overline{[\![t]\!]}$,
- for $\circ \in \{;, \sqcap, \sqcup\}$ we let $[\![t \circ u]\!] :\equiv [\![t]\!] \circ [\![u]\!]$,
- if c is an n-ary relation term constructor, then

$$[\![c(t_1, \ldots, t_n)]\!] :\equiv c([\![t_1]\!], \ldots, [\![t_n]\!]) \ ,$$

- for every pointed diagram $(G, \mathcal{I})$ we let $[\![(G, \mathcal{I})]\!] :\equiv [\![G_{\lceil \mathcal{I} \rceil}]\!]$.

We extend the concept to arbitrary hierarchical diagrams by letting the **vector interface** for G, written $\mathcal{V}_G$, be defined as the interface consisting of all nodes of G and one hyperedge with empty source and all nodes as targets, and we let the unfolding of an arbitrary diagram be the unfolding along its vector interface: $[\![G]\!] :\equiv [\![(G, \mathcal{V}_G)]\!]$. □

Actually, in the definition 3.8 of the readout, the first factor of the intersection exactly corresponds to $[\![G]\!]^{\smile}$, a fact that we shall use below for Theorem 4.3.

The substitution concept for relational diagrams need not be adapted at all, and the homomorphism concept can be carried over, too, without any adjustment. One might have the impression that it would be sufficient to have sub-diagrams at deeper levels in the hierarchy only related with homomorphisms instead of with substitutions, but since this would make strong assumptions about the monotonicity of the operators involved in constructing relation terms containing such sub-diagrams, which are obviously violated already by negation, we cannot in general allow this. So we use the homomorphism definition 3.4 without any change.

The importance of the unfolding of diagrams as defined via the vector interface lies in the following fact:

Theorem 4.3 For every two hierarchical relation diagrams G and H with a plain homomorphism k in-between, if $\mathcal{R} \vdash [\![G]\!] = [\![(H, \mathcal{V}_G ; k)]\!]$, then for every interface $\mathcal{I}$ for G we have $\mathcal{R} \vdash [\![(G, \mathcal{I})]\!] = [\![(H, \mathcal{I} ; k)]\!]$.

Proof: The proof relies on the fact that $\mathcal{I}$ corresponds to two projections $\pi_{\mathcal{I}}$ and $\rho_{\mathcal{I}}$ from the all-nodes-type to the respective input and output types of $\mathcal{I}$. Then the central line of reasoning can sloppily be summarised as follows (let $\mathcal{I}' := \mathcal{I} ; k$):

$$\begin{aligned} \mathcal{R} \vdash [\![(G,\mathcal{I})]\!] &= (\pi_{\mathcal{I}}^{\smile} \sqcap \mathbb{T} ; [\![G]\!]) ; \rho_{\mathcal{I}} = (\pi_{\mathcal{I}'}^{\smile} \sqcap \mathbb{T} ; [\![(H, \mathcal{V}_G ; k)]\!]) ; \rho_{\mathcal{I}'} \\ &= [\![(H, \mathcal{I}')]\!] = [\![(H, \mathcal{I} ; k)]\!] \end{aligned}$$ □

In simple cases, the hierarchical structure introduced above can be flattened immediately:

Definition 4.4 Let a relational diagram G with an edge e labelled with the pointed diagram $(H, (I, j))$ be given, and let $a_{\mathcal{I}}$ be the single hyperedge of I.

$$\begin{tikzcd} I \arrow[d, "k"'] & I^0 \arrow[l, "i"'] \arrow[r, "j"] \arrow[d, "i'"'] & H \arrow[d, "l"'] \\ G & G' \arrow[l, "r"'] \arrow[r, "s"] & G'' \end{tikzcd}$$

Let G' be the pushout-complement of the embedding i of I^0 into I and of the injective homomorphism k from I to G mapping $a_{\mathcal{I}}$ onto e. G' is just G without the edge e, and $r_{\mathbf{n}}$ is therefore the identity on nodes, and $r_{\mathbf{e}}$ is a natural injection on edges.

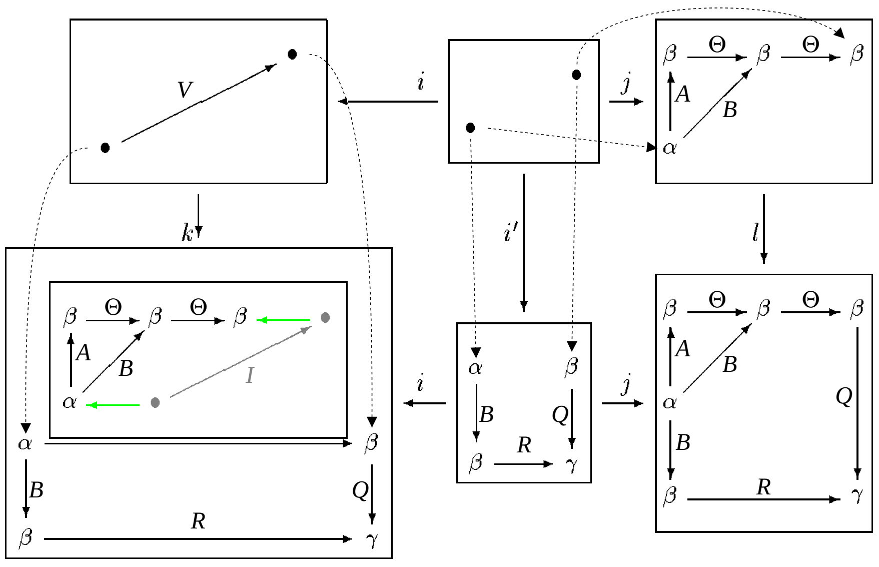
The **direct flattening** G'' of e is then the pushout of j and of the morphism i' from I^0 to G' resulting from the pushout-complement construction above.

In one word, G'' results from G by a double-pushout rewriting step with the rule $I \xleftarrow{i} I^0 \xrightarrow{j} H$ and with the matching k. The **direct flattening pair** $f := (f_{\mathbf{n}}, f_{\mathbf{e}})$ for G and e consists of $f_{\mathbf{n}} := r_{\mathbf{n}}^{\smile} ; s_{\mathbf{n}}$, which is a total function from the nodes of G to those of G'', and $f_{\mathsf{e}} := r_{\mathsf{e}}^{\smile} ; s_{\mathsf{e}} \sqcup \{e\} \times \{a : \mathcal{E}_H \bullet l_{\mathsf{e}}(a)\}$, which is a relation relating edges in G apart from e to the corresponding edges in G'' and relating e to all edges in the flattened image of H.

A direct flattening is called **trivial** if f_{e} is a function, too. □

The double-pushout rewriting step that is so-to-speak "remembered" in edges that are labelled with pointed diagrams makes our hierarchic diagrams quite similar to the multi-level graphs of [15]. There arbitrary interfaces are possible, while here we have restricted the interfaces to single hyperedges. The advantage of our approach, however, is that the application morphisms for the rules need not be remembered separately since they are "built-in" into the labelling.

Below we show a full-blown example of the double-pushout rewriting step in a direct flattening:

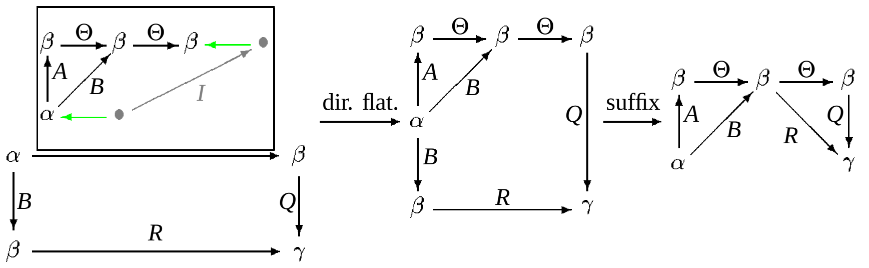


Definition 4.5 A diagram G_1 is a **general flattening** of a diagram G_2, if there is a diagram G_2' resulting from G_2 by a series of direct non-trivial flattenings, such that there is a surjective plain homomorphism s from G_2' to G_1. The composition $f_1 ; \cdots ; f_k ; s$ of the individual direct flattening pairs $f_1, \ldots, f_k$ with s is called the **flattening pair** for the general flattening of G_2 to G_1; the surjective plain homomorphism s is also called the **suffix** of the general flattening.

G_2 is also called a **general unflattening** of G_1. □

The rôle of s here is to make sharing of previously hierarchical parts possible. More precisely, it allows to use parts of the original graph G_1 in more than one place in an unflattening G_2, for example once in its top-level structure and one or more times in lower levels.

In the following example, after the direct flattening from above a suffix homomorphism identifies the two B-edges:



While any sequence of direct flattenings preserves semantics, a general flattening needs not do so because of the identifications of nodes resulting from the overlay s tucked onto the end. Nevertheless, we shall have a use for general flattenings.

Definition 4.6 A **relational matching** from a relational diagram G to another diagram H is a pair $f = (f_{\mathbf{n}}, f_{\mathbf{e}})$ consisting of a total function $f_{\mathbf{n}} : \mathcal{N}_G \to \mathcal{N}_H$ between the node sets and a relation $f_{\mathbf{e}} : \mathcal{E}_G \leftrightarrow \mathcal{E}_H$ between the edge sets of G and H, such that there exist a general unflattening H' of H with flattening pair h and a simple homomorphism g from G to H' with $f = g \mathbin{;} h$.

If H' is the limit of all such factorisations, then (g, h) is called the **canonical factorisation** of F. □

$$\begin{tikzcd}
G \arrow[dd, "f"'] \arrow[dr, "g"] \arrow[drr, "g'"] & & \\
& H' \arrow[dl, "h"] & H'' \arrow[l, "!u"'] \arrow[dll, "h'"] \\
H & &
\end{tikzcd}$$

To be precise, for this limit to exist we have to restrict the flattenings considered to those that can be regarded as "induced" by the edge relation $f_{\mathbf{e}}$.

We now shall use this canonical factorisation of relational matchings for defining a rewrite step for a functional rule (as before) with a relational matching from the left-hand side into the application diagram:

Definition 4.7 A **relational rewrite step** for a rule $(L \xrightarrow{r} R)$ and a relational diagram G together with a relational matching f from L to G is constructed as follows: Let (f', k) be the canonical factorisation of f via G', and let H' be the result of the simple rewrite step for $(L \xrightarrow{r} R)$ and f' with rewrite morphism s. Let $k = k'; q$ the factorisation of k into direct flattenings $k' : G' \to G''$ and the final overlay $q : G'' \to G$, then construct H'' as the result of transferring the direct flattenings k' to H' — this means to re-flatten the redex — and finally construct H, the result of the rewriting step, as the pushout of q and the appropriately transferred rewrite morphism s' — this means to re-introduce the sharing between intermediate sub-diagrams and other nodes. □

$$\begin{tikzcd}
{} \arrow[d, phantom, "\text{matching homomorphism}"] & L \arrow[r, "r"] \arrow[d, "f'"'] \arrow[ddd, bend right=60, "f"'] \arrow[dr, phantom, "\text{pushout}"] & R \arrow[d] \\
{} \arrow[d, phantom, "\text{direct flattenings}"] & G' \arrow[r, "s"] \arrow[d, "k'"'] \arrow[dr, phantom, "\text{transfer}"] & H' \arrow[d] \\
{} \arrow[d, phantom, "\text{plain surjective suffix}"] & G'' \arrow[r, "s'"] \arrow[d, "q"'] \arrow[dr, phantom, "\text{pushout}"] & H'' \arrow[d] \\
{} & G \arrow[r, "s''"] & H
\end{tikzcd}$$

Correctness of this rewriting step relies essentially on the fact that rules are just plain homomorphisms and rule application via total single pushouts can only add and identify items, but never delete items. Nevertheless it is interesting that this rewrite step is still correct although there are two overlays involved — the crucial point is that exactly the original sharing is re-established by the final pushout.

Theorem 4.8 In the relational rewriting step described above, we have

$$\mathcal{R} \vdash [\![G]\!] = [\![(H, \mathcal{V}_G ; s'')]\!] \ .$$

Proof: The key to the proof lies in the fact that the unfolding of the diagram G (with all the sharing) can be written in the shape of an intersection, $\mathcal{R} \vdash [\![G]\!] = L' \sqcap C$, where L' corresponds to an instance of the unfolding of the rule's left-hand side and C is the remaining context. We have for a suitable product with projections π and ρ that

$$\mathcal{R} \vdash L' \sqcap C = (L' ; \pi\breve{} \sqcap C ; \rho\breve{} \sqcap \mathbb{T} ; (\pi\breve{} \sqcap \rho\breve{})) ; \pi \ .$$

For the unfolding of the unshared diagram G'' we then have (informally and simplifyingly assuming that the vector interfaces make all the unfoldings compatible)

$$\mathcal{R} \vdash [\![G'']\!] = [\![G']\!] = (L' ; \pi\breve{} \sqcap C ; \rho\breve{}) ; \pi$$

(for the sake of simplicity we also assume an unsharing tearing off the image of L at all nodes). The conventional rewrite step uses $\mathcal{R} \vdash L = R$ for obtaining $\mathcal{R} \vdash L' = R'$ and therewith:

$$\mathcal{R} \vdash [\![G'']\!] = [\![G']\!] = (L' ; \pi\breve{} \sqcap C ; \rho\breve{}) ; \pi = (R' ; \pi\breve{} \sqcap C ; \rho\breve{}) ; \pi = [\![H']\!] = [\![H'']\!]$$

The construction of H from H'' via reinstating the sharing lets the unfolding of H be (still with the simplifying assumption about the vector interfaces)

$$\mathcal{R} \vdash [\![(H, \mathcal{V}_G ; s'')]\!] = (R' ; \pi\breve{} \sqcap C ; \rho\breve{} \sqcap \mathbb{T} ; (\pi\breve{} \sqcap \rho\breve{})) ; \pi = R' \sqcap C \quad .$$

Thus $\mathcal{R} \vdash L' = R'$ also justifies the whole relational rewrite step. $\square$

In the usual case it will probably useful to flatten any newly created flattenable hyperedges in the final rewriting result — this obviously does not influence the correctness. Only then we would obtain the example rewriting step in 2.2 directly as a relational rewriting step.

This formalisation of a relational rewrite step may at first look somewhat involved, but its advantage is that we did not have to introduce many new concepts in order to arrive at this formalisation. It is also pretty close to the steps that would have to be taken by an interactive proof assistant anyway.

5. Outlook and Conclusion

In this paper, we have extended the graphical calculus of relations in the shape introduced in [10] to the ability to cope with relational matchings for applying rules, or equivalently, to the possibility to have variable edge labels in rule sides match to whole sub-diagrams. This brings about a considerable shortening and simplification of proofs by hiding the technicalities. Thus a stronger emphasis on the key steps of a proof is reached.

We have so far only considered rather simple unflattening for the purpose of making rules applicable — just enough to yield a reasonable concept of derivations with relational matching.

It would of course be interesting what other operations — in a loose analogy to for example AC-matching — would make sense in order to get an even more general grip on derivation steps, and how that would bring us closer to the goal of freeing proofs from technicalities and concentrate on the essential ideas instead.

Another question that arises in this context is how to present proofs that have been arrived at in such a powerful graphical calculus. For this, and also for making the procedure of finding proofs more reliable, some tool support would be extremely helpful.

References

1. R. BANACH. *Term Graph Rewriting and Garbage Collection using Opfibrations*. Theoretical Computer Science **131** 29–94, 1994.

2. C. BRINK, W. KAHL, G. SCHMIDT, eds. *Relational Methods in Computer Science*. Advances in Computing. Springer-Verlag, Wien, New York, 1997.

3. C. BROWN, G. HUTTON. *Categories, Allegories and Circuit Design*. In: Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science, pp. 372–381, Paris, France, 1994. IEEE Computer Society Press.
4. S. CURTIS, G. LOWE. *A Graphical Calculus*. In B. MÖLLER, ed., Mathematics of Program Construction, Third International Conference, MPC '95, Kloster Irsee, Germany, July 1995, LNCS **947**, pp. 214–231. Springer Verlag, 1995.
5. H. EHRIG, M. KORFF, M. LÖWE. *Tutorial Introduction to the Algebraic Approach of Graph Grammars Based on Double and Single Pushouts*. In [6], pp. 24–37.
6. H. EHRIG, H.-J. KREOWSKI, G. ROZENBERG, eds. *Graph-Grammars and Their Application to Computer Science, 4th International Workshop*, Lecture Notes in Computer Science **532**, Bremen, Germany, 1990. Springer-Verlag.
7. P. J. FREYD, A. SCEDROV. *Categories, Allegories*, North-Holland Mathematical Library **39**. North-Holland, Amsterdam, 1990.
8. W. KAHL. *Kategorien von Termgraphen mit gebundenen Variablen*. Technischer Bericht 9503, Fakultät für Informatik, Universität der Bundeswehr München, 1995.
9. W. KAHL. *Algebraische Termgraphersetzung mit gebundenen Variablen*. Reihe Informatik. Herbert Utz Verlag Wissenschaft, München, 1996. ISBN 3-931327-60-4; also doctoral dissertation at Fakultät für Informatik, Universität der Bundeswehr München.
10. W. KAHL. *Algebraic Graph Derivations for Graphical Calculi*. In F. D'AMORE, P. G. FRANCIOSA, A. MARCHETTI-SPACCAMELA, eds., Graph Theoretic Concepts in Computer Science, 22nd International Workshop, WG '96, Caddenabbia, Italy, June 1996, Proceedings, LNCS **1197**, pp. 224–238. Springer-Verlag, 1997.
11. W. KAHL. *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, 1997.
12. W. KAHL. *Relational Treatment of Term Graphs With Bound Variables*. Journal of the IGPL **6** 259–303, 1998.
13. R. KENNAWAY. *Graph Rewriting in Some Categories of Partial Morphisms*. In [6], pp. 490–504.
14. M. LÖWE. *Algebraic Approach to Graph transformation Based on Single Pushout Derivations*. Technical Report 90/05, TU Berlin, 1990.
15. F. PARISI-PRESICCE, G. PIERSANTI. *Multilevel Graph Grammars*. In E. W. MAYR, G. SCHMIDT, G. TINHOFER, eds., Proc. 20th International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS **903**. Springer-Verlag, 1995.
16. G. SCHMIDT, T. STRÖHLEIN. *Relations and Graphs, Discrete Mathematics for Computer Scientists*. EATCS-Monographs on Theoretical Computer Science. Springer Verlag, 1993.
17. J. M. SPIVEY. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.
18. H. ZIERER. *Relation-Algebraic Domain Constructions*. Theoretical Computer Science **87** 163–188, 1991.