Algebraic Graph Derivations for Graphical Calculi

WOLFRAM KAHL

Department of Computing Science, German Armed Forces University Munich
e-mail: kahl@informatik.unibw-muenchen.de

1 Introduction

Relational formalisations can be very concise and precise and can allow short, calculational proofs under certain circumstances. Examples are can be found in [SS93], and also in the formalisation of second-order term graph rewriting in [Kah95b, Kah96]; for further applications of relational methods see also the book [BKS97].

In situations corresponding to the simultaneous use of many variables in predicate logic, however, either a style using predicate logic 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 [Kah95b, Kah96] we have been forced to employ both methods extensively, and, independently of other approaches, have been driven to develop a *graphical calculus* for making complex relation algebraic proofs more accessible.

It turns out that, although our approach shares many common points with those presented in the literature [BH94, CL95], it still is more general and more flexible than those approaches since we draw heavily on additional background in algebraic graph rewriting (see [EKL90] for a tutorial overview).

The part of the structure of relation algebra that can readily be exploited in graphical calculi is that of a *unitary pretabular allegory* (UPA, introduced in [FS90]). Allegories are a generalisation of categories to cope with relation-like structures; we shall not need any allegory theory in this paper, but only refer to it for comparison with one of the main streams of related work in the literature. In [BH94], an approach to transformations of expressions in UPAs via transformations of graphs has been presented and proven correct. The approach has been developed with a bias towards VLSI circuit development and the formalisation and drawings reflect this.

More or less building on the approach of [BH94], another approach to graphical calculi has been presented in [CL95], where a gentler introduction is given and an attempt is made to somewhat generalise beyond UPAs.

Both approaches, however, present the transformation rules as low-level graph manipulation rules and do not resort to any established graph transformation mechanism. As a result, there is only a fixed set of transformation rules that

correspond to the basic axioms of the calculus, but no general mechanism to formulate new rules corresponding to proven theorems or special definitions.

In this paper we start from a slightly more general definition of *diagram* as basic data structure for our graphical calculus, and we proceed to give algebraic definitions of rule application and derivation. We cleanly separate the syntax and the semantics of our diagrams and we define correctness of rules on a high level.

For reasons of space we do not present any proofs, but concentrate on giving ample motivation and at least a few examples. I gratefully acknowledge the comments of an anonymous referee.

2 Type and Relation Terms

The structure we are going to exploit in our diagram proofs is that of a locally complete unitary pretabular allegory (LCUPA) [FS90], which is essentially an abstract relation algebra in the sense of [SS93] (without negation), equipped with all direct products (which are understood to be formed in the underlying category of total functions throughout this paper).

For improving understandability we shall use the more widespread nomenclature of abstract relation algebra (rather than that of LCUPAs) and also its notation as agreed upon in [BKS97]. So 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 $R^{\smile} : B \leftrightarrow A$; **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 $\top_{A,B}$, and the **empty relation** is $\bot_{A,B}$. 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 **direct 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^*. Two sequences s and t can be *concatenated* to form the sequence $s\,\hat{}\,t$. For a function $f : X \to Y$, we denote the mapping of f to sequences by $f^* : X^* \to Y^*$.

A sequence of objects of a LCUPA is understood to be the corresponding finite product.

We write set-comprehensions according to the Z-notation [Spi89], which uses the pattern "{ *signature* | *predicate* • *term* }" instead of the otherwise frequently observed pattern "{ *term* | *predicate* }". So we can write, as an example, the set containing the first four square numbers as $\{n : \mathbb{N} \mid n < 4 \bullet n^2\} = \{0, 1, 4, 9\}$.

We now 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 2.1 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 in LCUPAs, we rather treat it separately.

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

Definition 2.2 A **relation term** of type $A \leftrightarrow B$ for two type terms A and B can be
- a *relation constant*, including $\mathbb{I}$ (if $A \equiv B$), $\mathbb{T}$, $\perp\!\!\!\perp$, 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 $B \leftrightarrow A$ (also written $R : B \leftrightarrow A$),
- 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 partial function with finite domain 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 Relational Diagrams

3.1 Syntax

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 3.1 A **relational diagram** is a labelled directed (hyper-)graph $(\mathcal{N}, \mathcal{E}, \mathbf{s}, \mathbf{t}, \mathbf{n}, \mathbf{e})$ with $\mathcal{N}$ its node sets, $\mathcal{E}$ its edge set, $\mathbf{s} : \mathcal{E} \to \mathcal{N}$ (resp. $\mathbf{s} : \mathcal{E} \to \mathcal{N}^*$) the source mapping, $\mathbf{t} : \mathcal{E} \to \mathcal{N}$ (resp. $\mathbf{t} : \mathcal{E} \to \mathcal{N}^*$) the target mapping, $\mathbf{n}$ 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. $\mathbf{n}^*(\mathbf{s}(e)) \leftrightarrow \mathbf{n}^*(\mathbf{t}(e))$). □

Homomorphisms between relational diagrams are defined as usual:

Definition 3.2 A **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 for all nodes v we have
$$\mathbf{n}_2(f_{\mathbf{n}}(v)) \equiv \tau(\mathbf{n}_1(v)) \quad ,$$
- there is a relation substitution σ such that for all edges a we have
$$\mathbf{e}_2(f_{\mathbf{e}}(a)) \equiv \sigma(\mathbf{e}_1(a)) \quad .$$

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

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: $G^0 := (\mathcal{N}, \emptyset, \emptyset, \emptyset, \mathbf{n}, \emptyset)$.

In contrast with the graphs of [CL95], which are equipped with a designated source and a designated target node, and with the pictures and networks of [BH94], which use a connection mechanism that is also based on a source-target view, but includes the possibility of considering multiple ports, our diagrams are not equipped with any such indication of direction.

Therefore, when considering the semantics of a diagram, a direction has to be imposed from the outside, and we use *interfaces* for this purpose.

In the simple graph case, an interface essentially is just a graph with two nodes and one variable-labelled edge inbetween together with a homomorphism that essentially just serves to flag a source node and a target node in the diagram in question. The general definition that also copes with hyperedges has a more complicated formulation:

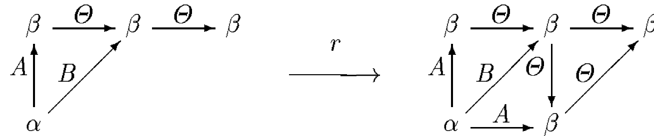
Definition 3.3 An **interface** (I, j) for a relational diagram G consists of a relational diagram I and a homomorphism j from I^0 to G, where I has only one edge ($|\mathcal{E}_I| = 1$) that is in addition labelled with a variable, and I does not have any isolated nodes. □

j could be considered as a restricted kind of partial morphism from I to G. For graph rules, we have an inclusion semantics "$L \sqsubseteq R$" in mind. Since it is usually advisable to preserve previously established information, we can arrive at a useful rule concept by just asking for a homomorphism between the rule sides. This homomorphism has to be plain since otherwise there could be clashes in the instantiation of variables:

Definition 3.4 A **rule** $(L \xrightarrow{r} R)$ consists of two relational diagrams L and R together with a plain homomorphism r from L to R. □

An example rule is draw in the following diagram; here α and β are type variables, Θ syntactically is a constant (an arbitrary equivalence relation) and P and Q are relation variables; the rule then reads, that if $\Theta : \beta \leftrightarrow \beta$, for all $A : \alpha \leftrightarrow \beta$ and $B : \alpha \leftrightarrow \beta$ the following inclusion holds:[1]

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



According to the definition of rules as single homomorphisms, rewriting will be defined by a single pushout construction[2] — the difference to the single-pushout approach of [Ken90, Löw90] is that here we still consider *total* homomorphisms:

Definition 3.5 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

$$\begin{array}{ccc} L & \xrightarrow{r} & R \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ G & \xrightarrow{s} & H \end{array}$$

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

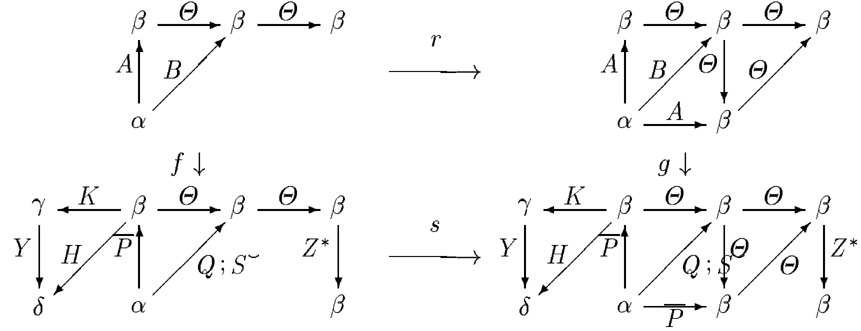
$$\begin{array}{ccc} L_i & \xrightarrow{r_i} & R_i \\ {\scriptstyle f_i}\downarrow & & \downarrow{\scriptstyle g_i} \\ G_{i-1} & \xrightarrow{s_i} & G_i \end{array}$$

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

[1] Actually, the diagram 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 everyobject 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 [EKL90].

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



3.2 Semantics

For abbreviating the formal treatment, we informally treat the product type constructor as associative, and 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 3.6 (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 $\langle m_1, \ldots, m_{\mathrm{ex}} \rangle$ be a sequentialisation of the nodes of G outside the range of j (this could be made uniquely determined by for example demanding a total ordering on the node set).

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:

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

Furthermore, let $\pi_{\mathrm{in}} : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}})))$ and $\pi_{\mathrm{out}} : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}^*(j^*(\mathbf{t}(a_{\mathcal{I}})))$ be the projections onto the source and target of the interface image. Let for every node $x \in \mathcal{N}_I$ of the interface diagram $\pi_x : \mathcal{T}_{G,\mathcal{I}} \to \mathbf{n}(j(x))$ be the projection onto its component, 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. (Since j need not be injective, there can be a choice of projections for π_s, but as we shall see by the construction below, this does not influence the result significantly. We can choose a canonic construction that assigns to $z \in j(\mathcal{N}_I)$ the projection corresponding to that node $x \in \mathcal{N}_I$ that is the first one in the sequence $\mathbf{s}(a_{\mathcal{I}})\hat{\ }\mathbf{t}(a_{\mathcal{I}})$, for which $z = j(x)$.)

The **readout** of G via $\mathcal{I}$ is now defined to be the relation term

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

with (see example and explanation below):

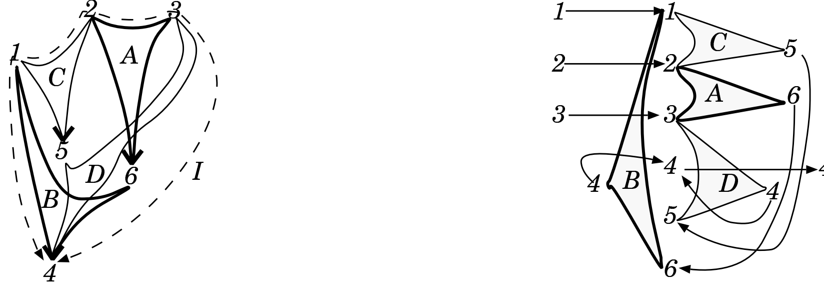
$$\begin{aligned} G_{\lceil \mathcal{I} \rceil} :\equiv \pi_{\mathrm{in}}^{\smile} ; \big(& \textstyle\bigsqcap \{ a : \mathcal{E} \bullet (\pi_{\mathbf{s}(a_{\mathcal{I}})} ; \mathrm{e}(a) \sqcap \pi_{\mathbf{t}(a_{\mathcal{I}})}) ; \mathbb{T} \} \\ & \sqcap \textstyle\bigsqcap \{ x, y : \mathcal{N}_I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

□

(The intersections over sets strictly speaking also present choices wrt. the construction of $G_{\lceil\mathcal{I}\rceil}$.)

What we have done here is to construct in a canonic way a relational expression that corresponds to that encoded in the graph — using the laws of relational calculus with products it can 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 should clarify the construction of $G_{\lceil\mathcal{I}\rceil}$. All simple edges there are labelled with the identity $\mathbb{I}$. Collapsing those edges returns the original diagram G, so the two are obviously equivalent.



On the other hand, the three main layers of nodes in the right diagram obviously correspond to the input type $\mathbf{n}^*(j^*(\mathbf{s}(a_{\mathcal{I}})))$, $\mathcal{T}_{G,\mathcal{I}}$ 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

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

This is equivalent to

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

and again (under the assumption of an appropriately nested input product and using the isomorphism $\mathsf{PAass} : \alpha\times(\beta\times\gamma)\to(\alpha\times\beta)\times\gamma)$ with:

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

which is easy to relate to the original diagram G.

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.

Unlike [CL95], we did not switch to predicate logic formulae, so we could stay inside the language of relation calculus extended with direct products (i.e., the language of LCUPAs).

Unlike [BH94], we started from a graph without additional hierarchic structure, so we had to construct $G_{\lceil\mathcal{I}\rceil}$ by “brute force”. But since we consider $G_{\lceil\mathcal{I}\rceil}$

to be only an intermediary result anyway, the artificialness of its structure does not hurt our approach at all.

A different approach would have been to use the algorithm proposed in [VH91] for transforming any predicate logic formula into a relational expression, but that algorithm has to be capable to deal with more general situations than those reflected in relational diagrams, and the result would have been similarly artificial in its nature anyway.

We now start considering an arbitrary model $\mathcal{R}$ for our formalism, that is, a relation algebra (or LCUPA) together with interpretations for all constants and term constructors. When we write $\mathcal{R} \models F$ for some relational formula F then that has to be taken to mean that for every valuation of type variables with objects of $\mathcal{R}$ and every valuation of relation variables with relations from $\mathcal{R}$ the semantics of F in $\mathcal{R}$ is true.

The central result about the readout construction then is that all the choices encountered there do not influence the semantics of the result:

Proposition 3.7 For every relational term X resulting from changing any choices made while constructing $G_{[\![\mathcal{I}]\!]}$, we have $\mathcal{R} \models G_{[\![\mathcal{I}]\!]} = X$. □

We also obtain the fact that plain homomorphisms can only decrease the semantics:

Lemma 3.8 For every interface (I, j) for K and every plain homomorphism k from K to G, the following holds: $\mathcal{R} \models G_{[\![(I,j;k)]\!]} \sqsubseteq K_{[\![(I,j)]\!]}$.

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

Based on the semantics, we can now form a general concept of admissibility of rules:

Definition 3.9 A rule $(L \xrightarrow{r} R)$ is **correct** if for all interfaces (I, j) for L,

$$\mathcal{R} \models L_{[\![(I,j)]\!]} = R_{[\![(I,j;r)]\!]} \quad .$$

□

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.

For all the specific rules listed in [BH94, CL95] corresponding diagram rules can be formulated, and the correctness proofs carry over for any $\mathcal{R}$.

Application of correct rules yields derivations with a useful semantics:

Proposition 3.10 Let an interface (I, j) for a relational diagram G and a rule $(L \xrightarrow{r} R)$ 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 $\mathcal{R} \models G_{\llbracket (I,j) \rrbracket} = H_{\llbracket (I,j\,;\,s) \rrbracket}$.

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

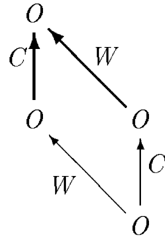
With all this, we can formulate a strategy for finding a proof of the inclusion formula $R \sqsubseteq S$ as a graph derivation on relational diagrams:

i) Construct a diagram G together with an interface (I, j) such that $\mathcal{R} \models R = G_{\llbracket (I,j) \rrbracket}$.
ii) Perform a suitable graph derivation on G, yielding H and the derivation morphism σ.
iii) Factorise σ into σ' from G to a suitable diagram H' and k from H' to H
iv) Recognise H' as a diagram with $\mathcal{R} \models S = H'_{\llbracket (I,j\,;\,\sigma') \rrbracket}$.

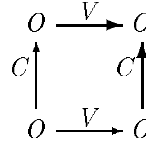
Only in rare cases the full derivation result H will be needed (yielding an equality), usually only an inclusion is required anyway.

3.3 Examples without Hyperedges

For our first example, a part of the proof of [Kah95b, 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:



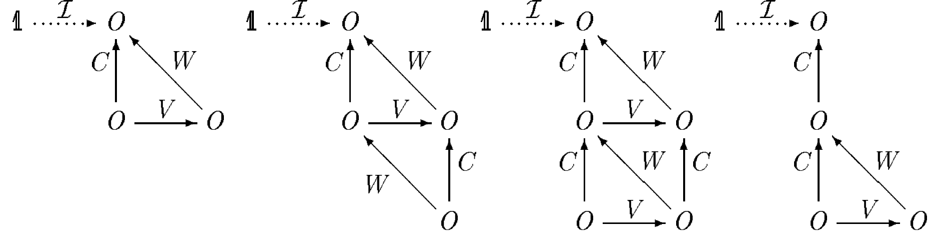
$W\,;C^{\smile} \sqsubseteq C\,;W^{\smile}$



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

For both rules we have drawn the right hand side; the left hand side is the subgraph 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 to the 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 wich interface into the left-hand side is considered.)

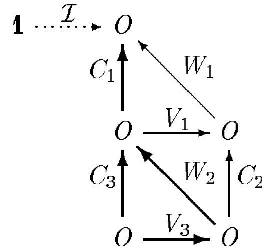
Now, for a derivation of $\mathbb{T};(C \sqcap V;W) \sqsubseteq \mathbb{T};(C \sqcap V;W);C$, first these two rules are applied in order and then Lemma 3.8:



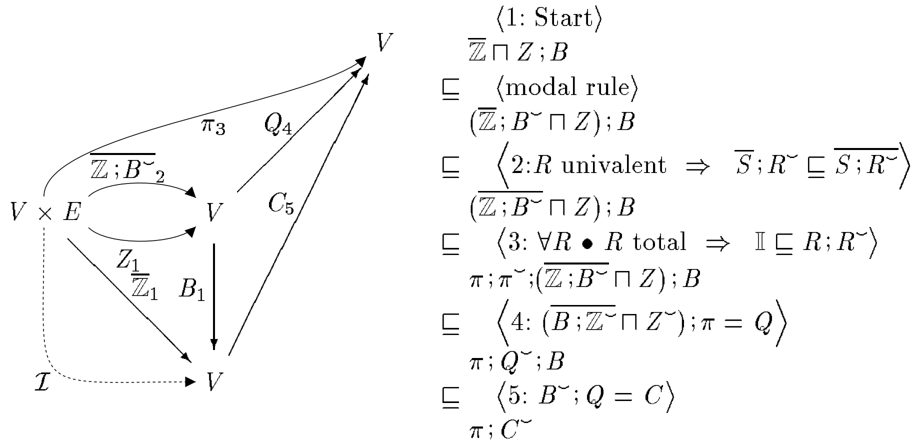
A standard inclusion chain for the same proof would be the following:

$\mathbb{T};(C \sqcap V;W)$	
$\sqsubseteq \mathbb{T};V;(W \sqcap V^{\smile};C)$	modal rule
$\sqsubseteq \mathbb{T};(V^{\smile} \sqcap W;C^{\smile});C$	$\mathbb{T};V \sqsubseteq \mathbb{T}$, modal rule
$\sqsubseteq \mathbb{T};(V^{\smile} \sqcap C;W^{\smile});C$	$W;C^{\smile} \sqsubseteq C;W^{\smile}$
$\sqsubseteq \mathbb{T};C^{\smile};(W \sqcap C;V^{\smile});C$	modal rule
$\sqsubseteq \mathbb{T};(W \sqcap V^{\smile};C);C$	$\mathbb{T};C^{\smile} \sqsubseteq \mathbb{T}$, $C;V^{\smile} \sqsubseteq V^{\smile};C$
$\sqsubseteq \mathbb{T};V^{\smile};(C \sqcap V;W);C$	modal rule
$\sqsubseteq \mathbb{T};(C \sqcap V;W);C$	$\mathbb{T};V^{\smile} \sqsubseteq \mathbb{T}$

A different way to present the diagrammatic proof could be via one graph with additional annotation of the edges with their "generation" (in addition, the edges needed in the result have been boldened):



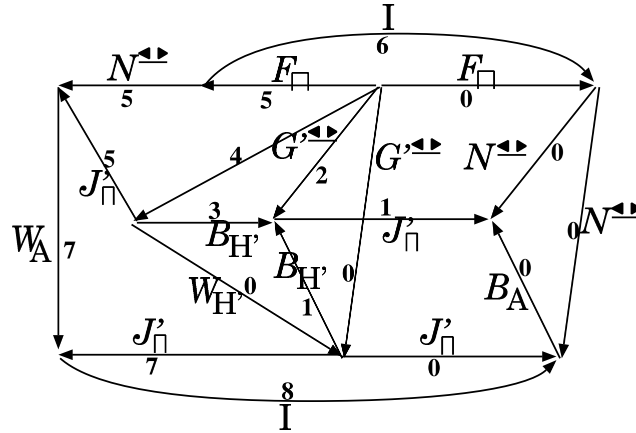
Obviously, the diagram proof is simpler and more intuitive than the linear (term) proof. The main reason for this are the frequent "changes of point of view" that are reflected in applications of modal rules or of the Dedekind rule. Not every proof, however, exhibits such a behaviour. For our second example consider a part of the proof of [Kah95b, Lemma 3.5.9]: ("$\overline{R}$" is the complement of "R"; since it does not play any part in the graphical part of the calculus, we omitted it in the introduction. From the point of view of Def. 2.2, the complement operator is just a unary term constructor.)



Here, only one "change of the point of view" was necessary. Therefore, the diagram proof has almost the same length as the linear proof.

When, however, in addition to changes of the point of view there are also references to many previously introduced nodes, then the linear proof would have to resort to heavy use of tuple constructions and manipulations, and the proof would become unreadable.

One example is a part of a proof that would be pretty hard to understand even in a mixed style using point variables in a predicate logic argument; it has been taken from [Kah95b, page 160] and uses quite a few special symbols and laws from the context there; these are however irrelevant for getting a general impression of the bandwith and graph size involved here, since it is these factors that make other approaches extremely hard to handle on such a problem:



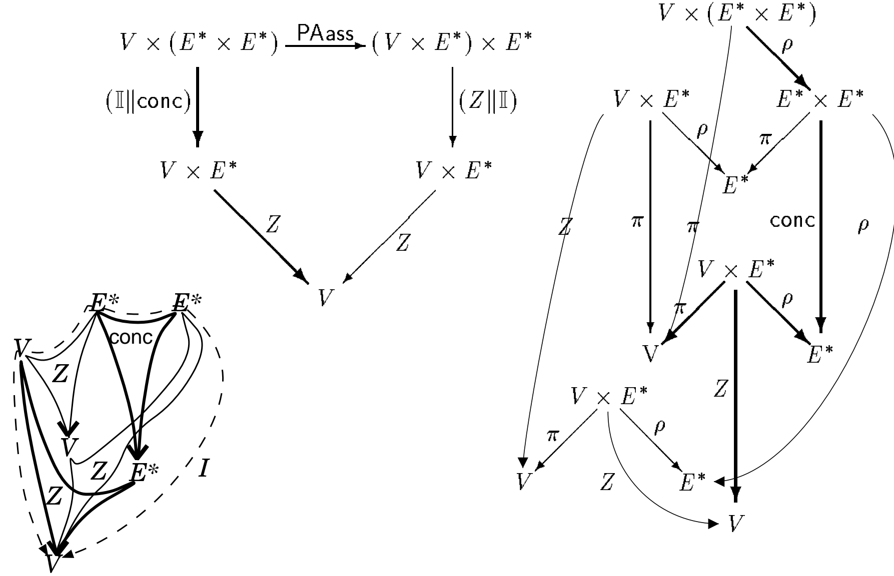
3.4 Example with Hyperedges

To see the beneficial effect the introduction of hyperedges can have, consider the following law (valid in the context of [Kah95b, Kah96]) which we want to use as

a rule:

$$(\mathbb{I}\|\mathsf{conc})\,;Z \sqsubseteq \mathsf{PAass}\,;(Z\|\mathbb{I})\,;Z$$

The middle diagram below directly depicts that rule:



On the left we have drawn a hypergraph diagram denoting the same law, and it obviously is far simpler and more intuitive.

Although the term rule and its immediate rendering as a diagram rule still look simple enough, the problem is that for applicability of this rule usually lots of auxiliary laws for tuple manipulation would be necessary. The same applies to making use of the transformation results.

When one tries to avoid this by expanding the definitions of $(_\|_)$ and PAass, the result is the diagram rule on the right, which is far more complicated than the hypergraph rule, although it presents exactly the same information.

4 Extension to Branching Derivations

So far, the only operations that have been reflected in the calculus are composition and intersection.

Although these are already the most useful, we still can extend our approach without too much effort to cover laws that have a union as the outermost constructor of the right-hand side. We call the corresponding rules *branching rules*, since they give rise to several branches inside one derivation.

Definition 4.1 A **branching rule** $\mathbf{R} = (L, (r_i, R_i)_{i \in \Gamma_{\mathbf{R}}})$ is a relational diagram L together with a $\Gamma_{\mathbf{R}}$-family of diagrams R_i with respective plain homomorphisms r_i from R_i to L. □

Every rule according to Def. 3.4 can be considered as a branching rule with a one-element index set, so our approach so far integrates smoothly with the extension.

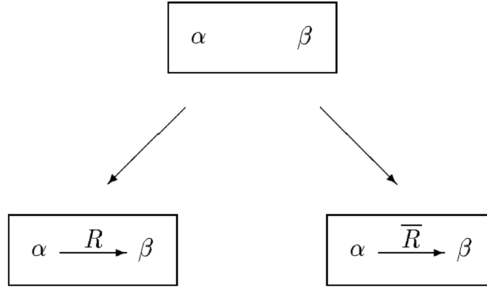
A **derivation tree** is then defined in the obvious way, with every edge resembling a single rewriting step from Def. 3.5, and the semantics carries over without any problems:

Definition 4.2 A branching rule is **correct**, if for all interfaces (I,j) for L, $\mathcal{R} \models L_{[\![(I,j)]\!]} = \bigsqcup\{i : \Gamma_{\mathbf{R}} \bullet R_{i[\![(I,j\,;r_i)]\!]}\}$. □

Proposition 4.3 For every derivation tree for G with leaves $(H_j)_{j \in \Gamma^*}$ and derivation morphisms s_j for the respective paths j defined accordingly, we have for any interface (I,j) of G that

$$\mathcal{R} \models K_{[\![(I,j)]\!]} = \bigsqcup\{j : \Gamma^* \bullet H_{j[\![(I,j\,;s_j)]\!]}\} \qquad \square$$

An example rule is the following, with type variables α and β and a relation variable R:



It corresponds, of course, to $\mathbb{T} \sqsubseteq R \sqcup \overline{R}$.

Even *joining rules* could be imagined, with union as outermost operator on the left-hand side, giving rise to derivations in the form of *directed acyclic graphs* (DAGs), but so far I have not yet encountered any useful examples for this.

5 Outlook and Conclusion

The rules we have considered could merge nodes through non-injective rule morphisms, but they could not delete any nodes or edges.

For this, the single-pushout approach we have presented here (albeit with total morphisms!) would have to be replaced with a double-pushout approach.

A nice application of this could be the restriction of derivations to rules that never increased the total number of nodes — the *DANGLING* condition in the gluing condition (see [Ehr78]) guarantees that with the applicability of such a rule the necessary nodes can in fact be deleted.

Restriction to three nodes would correspond to what is possible in the conventional relation calculus without resorting to direct products, and Roger Maddux has expressed interest in the class of theorems derivable with a "*bandwidth*" of at most four or five.

Summarising, we have seen that we can express

- **intersection** and **composition** inside relational diagrams,
- **products** via hyperedges and multiple nodes,
- **Dedekind** and **modal rules** via "changing point of view",
- **transitivity of inclusion** in derivations and
- **union** in branching rules.

Therefore, relational diagram proofs are useful whenever these concepts have to be used heavily, either explicitly in relational formulae or implicitly, as for example for products, that are used implicitly in predicate logic formulations with many variables.

The most important difference of our approach to those of [BH94] and [CL95] is that we have introduced an independent rule concept that can be used for arbitrary applications, and that we have exploited the categorical concepts of the algebraic approach to graph rewriting as the driving mechanism behind our derivation concept.

At the same time, we have created a rule mechanism that supports rule parameters and genericity in the typing of the rules at the same time — this is of course motivated by our work on typed term graph rewriting [Kah95a] in the context of the graphically interactive functional programming and program transformation system HOPS [ZSB86, Kah94, BK94].

References

[BH94] Carolyn Brown and Graham Hutton. Categories, allegories and circuit design. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 372–381, Paris, France, 4–7 July 1994. IEEE Computer Society Press.

[BK94] Arne Bayer and Wolfram Kahl. The **H**igher-**O**bject **P**rogramming **S**ystem "HOPS". In Bettina Buth and Rudolf Berghammer, editors, *Systems for Computer-Aided Specification, Development and Verification*, Bericht Nr. 9416, pages 154–171. Universität Kiel, 1994. URL: http://inf2-www.informatik.unibw-muenchen.de/HOPS/papers/Bayer-Kahl-94.ps.gz.

[BKS97] Chris Brink, Wolfram Kahl, and Gunther Schmidt, editors. *Relational Methods in Computer Science*. Advances in Computing. Springer-Verlag, Wien, New York, 1997. ISBN 3-211-82971-7.

[CL95] Sharon Curtis and Gavin Lowe. A graphical calculus. In Bernhard Möller, editor, *Mathematics of Program Construction, Third International Conference, MPC '95, Kloster Irsee, Germany, July 1995*, volume 947 of *LNCS*, pages 214–231. Springer Verlag, 1995.

[Ehr78] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science and Biology, International Workshop*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69, Bad Honnef, November 1978. Springer-Verlag.

[EKL90] Hartmut Ehrig, Martin Korff, and Michael Löwe. Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts. In Ehrig et al. [EKR90], pages 24–37.

[EKR90] Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Graph-Grammars and Their Application to Computer Science, 4th International Workshop*, volume 532 of *Lecture Notes in Computer Science*, Bremen, Germany, March 1990. Springer-Verlag.

[FS90] Peter J. Freyd and Andre Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1990.

[Kah94] Wolfram Kahl. Can functional programming be liberated from the applicative style? In Bjørn Pehrson and Imre Simon, editors, *Technology and Foundations, Information Processing '94, Proceedings of the IFIP 13th World Computer Congress, Hamburg, Germany, 28 August – 2 September 1994, Volume I*, volume A-51 of *IFIP Transactions*, pages 330–335. IFIP, North-Holland, 1994.

[Kah95a] Wolfram Kahl. Aspects of typed term graphs. In Tiziana Margaria, editor, *Kolloquium Programmiersprachen und Grundlagen der Programmierung, Adalbert Stifter Haus, Alt Reichenau, 11.–13. Oktober 1995*, Bericht MIP-9519, pages 104–109. Universität Passau, Fakultät für Mathematik und Informatik, December 1995.

[Kah95b] Wolfram Kahl. Kategorien von Termgraphen mit gebundenen Variablen. Technischer Bericht 9503, Fakultät für Informatik, Universität der Bundeswehr München, September 1995. 191 pages.

[Kah96] Wolfram 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.

[Ken90] Richard Kennaway. Graph rewriting in some categories of partial morphisms. In Ehrig et al. [EKR90], pages 490–504.

[Löw90] Michael Löwe. Algebraic approach to graph transformation based on single pushout derivations. Technical Report 90/05, TU Berlin, 1990.

[Spi89] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.

[SS93] Gunther Schmidt and Thomas Ströhlein. *Relations and Graphs, Discrete Mathematics for Computer Scientists*. EATCS-Monographs on Theoretical Computer Science. Springer Verlag, 1993.

[VH91] Paulo A. S. Veloso and Armando M. Haeberer. A finitary relational algebra for classical first-order logic. *Bulletin of the Section on Logic of the Polish Academy of Sciences*, 20(2):52–62, 1991.

[ZSB86] Hans Zierer, Gunther Schmidt, and Rudolf Berghammer. An interactive graphical manipulation system for higher objects based on relational algebra. In Gottfried Tinhofer and Gunther Schmidt, editors, *Proc. 12th International Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 246, pages 68–81, Bernried, Starnberger See, June 1986. Springer-Verlag.