Theoretical Computer Science 109 (1993) 181–224
Elsevier



# Algebraic approach to single-pushout graph transformation*

Michael Löwe

*FR 6-1, Technische Universität Berlin, Franklinstrasse 28/29, W-1000 Berlin 10, Germany*

*Abstract*

Löwe, M., Algebraic approach to single-pushout graph transformation, Theoretical Computer Science 109 (1993) 181–224.

The single-pushout approach to graph transformation interprets a double-pushout transformation rule of the classical algebraic approach which consists of two *total* graph morphisms as a single *partial* morphism from the left- to the right-hand side. The notion of a double-pushout diagram for the transformation process can then be substituted by a single-pushout diagram in an appropriate category of partial morphisms.

It can be shown that this kind of transformation generalizes the double-pushout framework. Hence, the classical approach can be seen as a special (and very important) case of the new concept. It can be reobtained from the single-pushout approach by imposing an application condition on the redices which formulates the gluing conditions in the new setting. On the other hand, single-pushout transformations are always possible even if the gluing conditions for the redex are violated.

The simpler structure of a direct transformation (one pushout diagram instead of two) simplifies many proofs. Hence, the whole theory for double-pushout transformations including sequential composition, parallel composition, and amalgamation can be reformulated and generalized in the new framework.

Some constructions provide new effects and properties which are discussed in detail.

## 1. Introduction

Graph grammars provide an intuitive description for the manipulation of complex graph-like structures as they occur in databases, operating systems, and complex applicative software. Besides that all approaches to graph transformation systems offer theoretical results which help in the analysis of such systems.

Especially the algebraic approach [8, 9, 12] has been worked out for several years now and provides results for parallelism analysis [25, 27], efficient evaluation of

*Correspondence to*: M. Löwe, FR 6-1, Technische Universität Berlin, Franklinstrasse 28/29, W-1000 Berlin 10, Germany.

*This work has been partly supported by the ESPRIT Basic Research Working Group No. 3299 "Computing by Graph Transformation".

functional expressions [33, 22], synchronization mechanisms [4], distributed systems [3, 10, 38], implementation of abstract data types [29], and context-free hyperedge replacement [18, 19].

A graph transformation rule $(L, K, R)$ conceptually consists of three graphs $L$, $K$, and $R$. $L$ is the left-hand side of the rule. It formulates the precondition under which the rule is applicable. $K$, in most cases a subgraph of $L$ and $R$, describes the part of the left-hand side which is going to be preserved by rule application. Thus, $L-K$ is the part which a rule application is going to delete and $R-K$ is added. Here, the intermediate graph $K$ gets a second role: It describes the context into which added components are going to be integrated. ($K$ is called "gluing graph".)

A rule is applicable to a graph $G$ if $G$ contains a homomorphic image of $L$. The application of a rule $(L, K, R)$ deletes all items in $G$ which correspond to objects in $L-K$ in the first step. It results in the so-called context graph $D$. Second, it adds all items in $R-K$ to $D$. The connection between "new" items in $R-K$ and "old" objects in $D$ is described by the relation of the "new" items in $R-K$ to objects in $K$. Thus, application of a rule $r=(L, K, R)$ to a graph $G$ consists of four steps:

(1) Try to find $L$ in $G$. If there are some images of $L$ in $G$ *choose* one and continue. Otherwise, $r$ is not applicable to $G$. [In some approaches, the matching phase includes the *check* of additional application conditions (see below).]

(2) *Remove* the part of $G$ which corresponds to $L-K$.

(3) *Add* $R-K$ to the result of the last step.

(4) *Embed* $R-K$ into $G-(L-K)$ as it is given by the corresponding relation between $R-K$ and $K$.

This series of four steps seems to be common to all approaches to graph transformation; cf. [26].[1] The algebraic approach to graph transformation (cf. Appendix A for basic notions) summarizes these four steps in a single categorical construction of a double-pushout diagram which facilitates many proofs that would be very hard to obtain on the more concrete, operational level: A rule is a pair $(l: K \to L, r: K \to R)$ of *total* graph morphisms and a direct transformation with the rule $(l: K \to L, r: K \to R)$ from a graph $G$ to a graph $H$ is possible if there is a context graph $D$ together with a gluing morphism $k: K \to D$ such that $G$ is the pushout of $l$ and $k$ and the graph $H$ is the pushout of $r$ and $k$ (for more details compare Appendix A). With these definitions, all operational effects of a direct transformation are encapsulated in a single categorical colimit construction and, therefore, all universal properties known for this construction within category theory are inherited [1, 21]. Thus, many proofs do not bother about operational details but only rely on abstract arguments about colimits.

Since all results about algebraic graph transformation require the rules' left-hand sides to be injective,[2] the rule concept can be simplified when the pair

[1] However, there are individual differences in each phase and the formulation of the embedding area by a subgraph $K$ of $L$ and $R$ is an idealization.

[2] With noninjective left-hand sides, the context graph in a transformation from $G$ to $H$ need not be unique.

$(l: K \to L, r: K \to R)$ of *total* morphisms is seen as a *partial* morphism $(r': L \to R)$ which is defined on $l(K)$ only and coincides with $r$ on its domain. Now the concept of direct transformation reduces to a single-pushout construction: $G$ transforms to $H$ using the rule $(r': L \to R)$ if there is a *total* matching morphism (or redex) $m: L \to G$ such that $H$ is the pushout of $r'$ and $m$ (here in the category of graphs and partial morphisms). It is this single-pushout concept which is comprehensively elaborated below. It turns out to be more general than the double-pushout framework and that all corresponding proofs are less complex due to the simpler underlying notion of direct transformation.

Single-pushout transformations in a setting of some sort of partial morphisms have been investigated in [36, 23].

Raoult [36] introduces two conceptually very different approaches. The first one is described in the category of sets and partial mappings. A rule is a partial morphism $r: L \to R$, i.e. a partial map which respects the graph structure[3] on all objects of $L$ it is defined for.[4] A redex $m: L \to G$ in some graph $G$ is a total morphism of this type. The result of applying $r$ at $m$ is constructed by two steps. First, the pushout $(H, r_m: G \to H, m_r: R \to H)$ of $r$ and $m$ in the category of sets and partial maps is built. In the second step, a graph structure is established on $H$ such that the pushout mappings $r_m$ and $m_r$ become morphisms. He characterizes the situations in which this graph structure uniquely exists; double-pushout transformations with their application conditions (cf. Appendix A) are special cases of these situations.

The second model of graph transformation in [36] uses another kind of partiality for the morphisms: a rule is a total map $r: L \to R$, which is only partially compatible with the graph structure. Let rewrite($r$) denote the set of objects which are not homomorphically mapped by $r$. A redex $m: L \to G$ is total which means now rewrite($m$) $= \emptyset$. Application of $r$ at $m$ is again defined by two steps. First construct the pushout $(H, r_m: G \to H, m_r: R \to H)$ of $r$ and $m$ in the category of sets and total mappings and second impose a graph structure on $H$ such that the pushout mappings become as compatible as possible, i.e. such that rewrite$(r_m) = m$(rewrite($r$)) and rewrite$(m_r) = r$(rewrite($m$)). Raoult [36] gives sufficient conditions for the unique existence of this structure. This approach has the major disadvantage that objects cannot be deleted at all (compare the intuitive graph transformation model above).

Kennaway [23] provides a categorical description for the second approach of [36]. Graphs are represented the same way. Morphisms $f: A \to B$ are pairs $(f, hom)$. The first component is a total mapping from $A$ to $B$. The second component provides a subset of $A$ on which $f$ respects the graph structure. A rule $r: L \to R$ is any morphism in this sense and a redex $m: L \to G$ is a total morphism which now means $hom_m = L$. He shows that under certain conditions the two-step construction of [36] coincides with the pushout construction in the category of graphs and the so-defined morphisms.

[3] The graph structure is imposed on a set by a successor relation, and a labeling function.

[4] We disregard the variable concept for this general discussion.

Unfortunately, only sufficient conditions for the existence of pushouts are given. Besides that, object deletion remains impossible.

The concept in [23] has been further developed in [17]. They introduce "generalized graph rewriting" which uses the same kind of graph morphism. The corresponding transformation concept not only involves a pushout construction but also a coequalizer. Since both construction are carried out in different categories (of total resp. partial morphisms), theoretical results are difficult to obtain.

The idea which is elaborated below is to resume the first approach in [36]. His concept of partial mappings which are compatible with the graph structure on their domain can be generalized to a concept of partial homomorphisms on special categories of algebras such that pushout construction in these categories is always possible. Hence, we get rid of any application conditions. If, however, the necessary and sufficient conditions of [36] are satisfied, the construction of pushout objects coincides with his two-step construction.[5]

Recently, Kennaway [24] independently started to study graph transformation in some categories of partial morphisms of this type. His work is based on the categorical formulation of a partial morphism provided by [37]. While we consider concrete algebraic categories, [24] stays in a purely categorical framework. Future research has to show how both approaches can benefit from each other.

Van den Broek [5] introduces another kind of single-pushout transformations based on "partial" morphisms. Partiality in this framework is described by total morphisms which map objects "outside their domain" to *marked* objects in their codomain.[6] Single pushout transformations with this type of morphisms corresponds to transformations in *junk*- or *sink*-completed structures described in Appendix B.

The article is organized as follows.[7] Section 2 provides the algebraic foundations for colimit constructions with partial morphisms. Especially we characterize the class of algebraic structures which has all finite colimits, so-called *graph structures*. Section 3 models graphs, hypergraphs, and other similar structures as graph structures and introduces the single-pushout transformation concept for all of these objects. It is shown that the single-pushout approach generalizes the double-pushout framework. A running example demonstrates the expressive power of the new concept. Sections 4, 5, and 6 are devoted to sequential composition, parallel composition, and amalgamation of single-pushout rules and transformations, respectively. They provide a comprehensive theory of rule composition. All properties that differ from the double-pushout case are discussed. The conclusion (Section 7) addresses some issues of further research.

[5] Actually, the whole theory presented in the following has been very much motivated and stimulated by the pushout constructions in the category of sets and partial mappings the author learned about by [36]. In this paper, these constructions are generalized to the level of algebras and partial homomorphisms.

[6] Marked objects indicate deleted or garbage items.

[7] The results presented in the following have been presented in [28] for the first time. The basic ideas of the single-pushout approach used here have been published in [30].

## 2. Partial morphisms and graph structures

This section provides a general introduction to colimit constructions in algebraic categories with partial homomorphisms.[8] The first central result provides necessary conditions for those categories to be closed w.r.t. colimits, namely that the signature contains unary operator symbols only. Signatures of this kind are called *graph structures*. The second main result shows that categories of graph structures and partial homomorphisms have all finite colimits. Both results characterize the structures which can be transformed by single-pushout constructions.[9] Some examples, how graphs, labeled graphs, hypergraphs, and more complex graph-like structures can be seen as graph structures, are given at the beginning of Section 3.

**Definition 2.1** (*Partial homomorphism*). If Sig is signature and $A$, $B$ are Sig-algebras, a *partial* Sig-*homomorphism* $h: A \to B$ is a total homomorphism from some subalgebra $A_h$ of $A$ to $B$. $A$ is the *domain*, $B$ the *codomain*, and $A_h$ the *scope* of $h$.

Since the scope $A_h$ of a partial homomorphism $h: A \to B$ is a subalgebra of $A$, we get $h(C) \subseteq B$ for each $C \subseteq A$ and, for each $D \subseteq B$, $h^{-1}(D) \subseteq A$.[10]

**Proposition 2.2** (Category of partial homomorphisms). *All* Sig-*algebras and all partial* Sig-*homomorphisms form a category* $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$.

**Proof.** The composition $f \circ g$ of two homomorphisms $g: A \to B$ and $f: B \to C$ is given by the componentwise composition of the underlying partial mappings. Its scope is $A_{f\,g} = g^{-1}(B_f \cap g(A_g))$. It is a subalgebra of $A$ since $B_f$ and $A_g$ are subalgebras of $B$ and $A$, respectively and $g(A_g)$ and $B_f \cap g(A_g)$ are subalgebras of $B$. That $f \circ g$ is homomorphic on its scope is implied by the fact that $(f \circ g)_{|A_{f\,g}} = f_{|B_f} \circ g_{|A_{f\,g}}$ which are total Sig-homomorphisms.[11] Composition of partial mappings is associative. The identities $id_A: A \to A$ for each algebra $A$ in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$ are provided by the corresponding total identity homomorphisms of Alg(Sig).[12] They satisfy for all partial homomorphisms $g: A \to B$ and $f: B \to A$, $id_A \circ f = f$ and $g \circ id_A = g$. □

Note that this definition of partial Sig-homomorphisms coincides with the usual category-theoretic definition in terms of subobjects and pullbacks as it can be found e.g. in [37].

[8] For basic notions and constructions of universal algebra compare [15].

[9] Recently, Ehrig et al. [11] have provided some results in this direction for the double-pushout approach.

[10] $\subseteq$ denotes the subalgebra relation, $h(C) = \{h(x) \mid x \in C\}$, and $h^{-1}(D) = \{x \mid h(x) \in D\}$.

[11] If $f: A \to B$ is a partial homomorphism, $C$ a subalgebra of $A$, and $D$ a subalgebra of $B$, $f_{|C}$ denotes the domain restriction of $f$ to $C$ and $f^{|D}$ denotes the codomain restriction of $f$ to $D$, i.e. the scope of $f^{|D}$ is given by $f^{-1}(D)$ and the definition of $f^{|D}$ coincides with the definition of $f$ on its scope.

[12] Alg(Sig) denotes the category of all Sig-algebras together with all *total* Sig-homomorphisms.

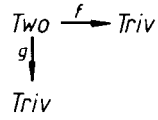


Diagram 1. Pushout situation for *Two* and *Triv*.

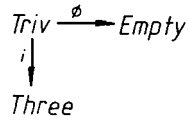


Diagram 2. Pushout situation for *Triv*, *Three*, and *Empty*.

Since we want to use $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ as a basis for graph transformation, we are mainly interested in pushout constructions in $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. Therefore, it has to be investigated under which conditions $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ has all pushouts.

**Proposition 2.3** (Pushout-incompleteness). $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ *is not closed w.r.t. pushouts if* Sig *contains constants or operator symbols with more than one argument.*

**Proof.** First, suppose $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ contains a constant $\mathsf{c}:\to\mathsf{cs}$. Consider Diagram 1 in which the Sig-algebras and homomorphisms are defined by:

(1) $Triv ::= Triv_s=\{*\}$ for all $s\in\mathsf{S}$ and $\mathsf{op}^{Triv}(*,\ldots,*)=*$ for all operators $\mathsf{op}:\mathsf{s}_1,\ldots,\mathsf{s}_n\to\mathsf{s}_{n+1}\in\mathsf{OP}$;[13]

(2) $Two ::= Two_{\mathsf{cs}}=\{*, a\}$, $Two_s=\{*\}$ for $s\neq\mathsf{cs}$, and $\mathsf{op}^{Two}(x_1,\ldots,x_n)=*$ for all operators $\mathsf{op}:\mathsf{s}_1,\ldots,\mathsf{s}_n\to\mathsf{s}_{n+1}\in\mathsf{OP}$;

(3) $g: Two\to Triv$ is the unique total homomorphism from *Two* to *Triv*; and

(4) $f: Two\to Triv$ is undefined for $a$ and $f(*)=*$ otherwise.

If there was an algebra $X$ and partial homomorphisms $f_g: Triv\to X$ and $g_f:\mathrm{Triv}\to X$ such that $g_f\circ f=f_g\circ g$, firstly $X_{\mathsf{cs}}\neq\emptyset$ because it must contain $\mathsf{c}^X$ and secondly $f_g(*)=f_g(\mathsf{c}^{Triv})=\mathsf{c}^X=g_f(\mathsf{c}^{Triv})=g_f(*)$ due to $f_g$ and $g_f$ being homomomorphic. This implies $f_g\circ g(a)=\mathsf{c}^X$. On the other hand, $g_f\circ f$ is undefined for $a$ since $f$ is. The arguments above lead to a contradiction to the assumption that there is a completion of Diagram 1 making it commute. Hence, there is no $X$, $f_g: Triv\to X$, and $g_f: Triv\to X$ with $g_f\circ f=f_g\circ g$ which implies that there is no pushout object for $f$ and $g$.

Second, suppose $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ contains no constants and at least one operator symbol $\mathsf{f}:\mathsf{fs}_1,\ldots,\mathsf{fs}_n\to\mathsf{fs}_{n+1}$ with $n\geq 2$. We construct a situation, depicted in Diagram 2, which cannot have a pushout completion. The participating algebras and homomorphisms are defined by:

(1) *Triv* is again the terminal algebra having $Triv_s=\{*\}$ for all $s\in\mathsf{S}$;

[13] *Triv* is the terminal object in Alg(Sig).

(2) since there are no constants, the empty algebra *Empty* is in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$: $Empty_s = \emptyset$ for all $s \in \mathsf{S}$ and $\mathsf{op}^{Empty} = \emptyset$ for all $\mathsf{op} \in \mathsf{OP}$;[14]

(3) *Three* is constructed as follows: for all $s \in \mathsf{S}$, $Three_s = \{*, a, b\}$ and for all operators $(\mathsf{op}: \mathsf{s}_1, \ldots, \mathsf{s}_n \to \mathsf{s}_{n+1}) \in \mathsf{OP}$, we define

$$\mathsf{op}^{Three}(x_1, \ldots, x_n) = \begin{cases} a & \text{if for all } i = 1, \ldots, n\text{: } x_i = a, \\ b & \text{if for all } i = 1, \ldots, n\text{: } x_i = b, \\ * & \text{otherwise;} \end{cases}$$

(4) $i: Triv \to Three$ is the inclusion of *Triv* in *Three*;

(5) $\emptyset: Triv \to Empty$ denotes the empty, everywhere undefined homomorphism.

The absence of constants guarantees that *Three* is well-defined.[15] If all carriers of *Three* are restricted to $\{a\}$ or to $\{b\}$, we obtain two subalgebras *Triv-one* and *Triv-two* and two partial homomorphisms $f_1: Three \to Triv\text{-}one$ and $f_2: Three \to Triv\text{-}two$ such that the scopes are given by $\mathrm{Three}_{f_1} = Triv\text{-}one$ and $Three_{f_2} = Triv\text{-}two$, respectively, and $f_1$ and $f_2$ are the identities on their scopes. Obviously, there are the unique partial homomorphisms $\emptyset: Empty \to Triv\text{-}one$ and $\emptyset: Empty \to Triv\text{-}two$ such that (1) and (2) in Diagram 3 commute.

Now assume the existence of pushouts and let $(X, i_\emptyset: Empty \to X, \emptyset_i: Three \to X)$ be the pushout of $\emptyset$ and $i$. Note that $i_\emptyset = \emptyset$ since it is the only partial homomorphism from *Empty* into some other algebra. Since (1) and (2) in Diagram 3 commute, there must be $u_1: X \to Triv\text{-}one$ and $u_2: X \to Triv\text{-}two$ such that (i) $u_1 \circ \emptyset_i = f_1$ and (ii) $u_2 \circ \emptyset_i = f_2$. (i) requires that the element $a$ in each carrier of *Three* is contained in $Three_{\emptyset_i}$ and (ii) requires that the element $b$ in each carrier of *Three* is an element of $Three_{\emptyset_i}$. Since the scope of $\emptyset_i$, i.e. $Three_{\emptyset_i}$, must be a subalgebra of *Three* (cf. Definition 2.1), $f^{Three}(a, b, a, \ldots, a) = * \in Three_{\emptyset_i}$. This results in $\emptyset_i \circ i(*)$ to be defined on the carriers for sort $\mathsf{fs}_{n+1}$ while $i_\emptyset \circ \emptyset = \emptyset \circ \emptyset = \emptyset$ is undefined everywhere. Hence, $i_\emptyset \circ \emptyset \neq \emptyset_i \circ i$ which is a contradiction to the assumption that $(X, i_\emptyset: Empty \to X, \emptyset_i: Three \to X)$ is the pushout object. Due to the fact that the contradiction occurs for each choice of possible pushout objects, there cannot be any. □

The negative result of Proposition 2.3 motivates the following definition. We distinguish signatures which contain monadic operator symbols only. The theory presented in the following is the theory of these so-called *graph structures*.

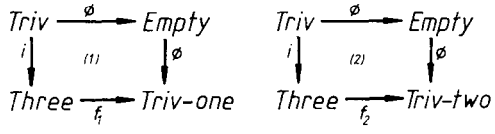


Diagram 3. Commuting diagrams for *Triv*, *Three*, and *Empty*.

[14] Note that due to the absence of constants, *Empty* is a subalgebra of each algebra in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$.

[15] Note that $\mathsf{op}^{Three}(x_1, \ldots, x_n)$ provides $*$ if some arguments are $a$ and some arguments are $b$.

**Definition 2.4** (*Graph structures*). A signature is a *graph structure* if it contains unary operator symbols only.

All terms w.r.t. a graph structure have a very special form:
(1) there are no ground terms due to the absence of constants;
(2) each term contains exactly one variable due to the absence of operators with more than one argument.
Thus, all terms represent derived unary operators. They can be sorted w.r.t. their value sort and the sort of the unique variable in them. Hence, we write $\mathsf{T}^{\mathsf{Sig}}_{s\to s'}$ for the following set of terms $\{t \mid t\in \mathsf{T}_{\mathsf{Sig},s'}(\{x\}),\ x\in X_s\}$. If $t\in\mathsf{T}^{\mathsf{Sig}}_{s\to s'}$, $x\in X_{\mathsf{s}}$ is the variable in $t$, $A$ is a Sig-algebra, and $a\in A_{\mathsf{s}}$, we write $t^A(a)$ for the evaluation of $t$ in $A$ using the variable assignment $x\mapsto a$.

**Lemma 2.5** (Subalgebras). *If* $\mathsf{Sig}=(\mathsf{S},\mathsf{OP})$ *is a graph structure and* $A$ *is a* Sig*-algebra, then the set of subalgebras of* $A$ *is closed w.r.t. intersection and union.*

**Proof.** Closure w.r.t. intersection is a general property for all signatures; cf. [15]. If $\mathscr{C}$ is a set of subalgebras of $A$, $\bigcup\mathscr{C}$ is also a subalgebra of $A$ if we define $\bigcup\mathscr{C}_s=\bigcup_{C\in\mathscr{C}}C_s$ for all $s\in\mathsf{S}$ and $\mathsf{op}^{\cup\mathscr{C}}=\bigcup_{C\in\mathscr{C}}\mathsf{op}^C$ for all $\mathsf{op}\in\mathsf{OP}$. Since all operators are unary, $\mathsf{op}^{\cup\mathscr{C}}:\bigcup\mathscr{C}_s\to\bigcup\mathscr{C}_{s'}$ is defined for all $x\in\bigcup\mathscr{C}_s$. It is well-defined because all $C\in\mathscr{C}$ are subalgebras of $A$. Hence, $\bigcup\mathscr{C}\subseteq A$. □

With Lemma 2.5, we immediately obtain the following result for arbitrary graph structures $\mathsf{Sig}=(\mathsf{S},\mathsf{OP})$: If $A$ is a Sig-algebra and $B=(B_s)_{s\in S}$ is a family of subsets of $A$, i.e. $(B_s\subseteq A_s)_{s\in S}$, then there is a greatest subalgebra of $A$ whose carriers are contained in $B$, namely $\bigcup\{C\subseteq A \mid C_s\subseteq B_s \text{ for all } s\in\mathsf{S}\}$. This implication of Lemma 2.5 is crucial for the following construction of pushouts in $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$.

**Construction 2.6** (*Pushouts in graph structures*). If $\mathsf{Sig}=(\mathsf{S},\mathsf{OP})$ is a graph structure and $f: A\to B$ and $g: A\to C$ is a pair of (partial) Sig-homomorphisms, the pushout $(D, f_g: C\to D, g_f: B\to D)$ of $f$ and $g$ in $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ can be constructed in four steps. (The pushout situation is depicted in Diagram 4.)
(1) Construction of the *gluing object* $f\triangledown g$ which is a subobject of $A$: $f\triangledown g$ is the largest subalgebra of $A$ which satisfies
(a) $f\triangledown g\subseteq A_f\cap A_g$ and
(b) for all $x\in f\triangledown g$ and $y\in A$, $f(x)=f(y)$ or $g(x)=g(y)$ implies $y\in f\triangledown g$.

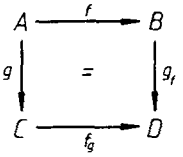


Diagram 4. Pushout situation.

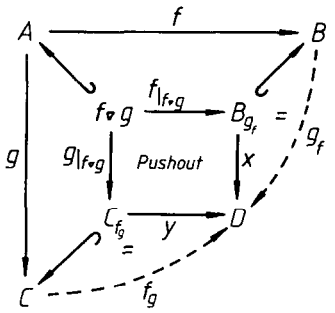


Diagram 5. Pushout construction for partial homomorphisms.

(2) Construction of the *scopes* of $f_g$ and $g_f$:

(a) The scope of $f_g$, i.e. $C_{f_g}$, is the largest subalgebra of $C$ whose carriers are contained in $(C-g(A))\cup g(f\triangledown g)$.

(*b*) Similarly, $B_{g_f}$ is the largest subalgebra of $B$ whose carriers are contained in $(B-f(A))\cup f(f\triangledown g)$.[16]

(3) *Gluing* construction of $D$: $D=(B_{g_f}+C_{f_g})/\sim$, where $x\sim y$ if there is an item $z\in f\triangledown g$ such that $x=f(z)$ and $y=g(z)$.[17]

(4) Construction of the *pushout homomorphisms*: $f_g: C\to D$ has the scope $C_{f_g}$ and is defined for all $x\in C_{f_g}$ by $f_g(x)=[x]_\sim$. Similarly, $g_f: B\to D$ is defined on its scope $B_{g_f}$.

Note that Construction 2.6 includes a pushout construction for total homomorphisms. The first two steps construct subalgebras of $A$, $B$, and $C$, i.e. $f\triangledown g$, $B_{g_f}$, and $C_{f_g}$, respectively, such that the domain restrictions of $f$ and $g$ w.r.t. $f\triangledown g$ are total homomorphisms $f_{|f\triangledown g}:(f\triangledown g)\to B_{g_f}$ and $g_{|f\triangledown g}:(f\triangledown g)\to C_{g_f}$, respectively. The object $D$, constructed in the third step, coincides with the pushout object of $f_{|f\triangledown g}$ and $g_{|f\triangledown g}$ in the category of Sig-algebras and total homorphisms. Also $f_g$ and $g_f$ coincide with the corresponding total pushout homomorphisms if they are restricted to their scopes. The whole situation is drawn in Diagram 5.

**Theorem 2.7** (Pushouts of graph structures). *If $f: A\to B$ and $g: A\to C$ is a pair of morphisms in a category of graph structures* $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$, *the object $D$ together with the morphisms $f_g: C\to D$ and $g_f: B\to D$ as they are constructed in Construction* 2.6 *is the pushout of $f$ and $g$ in* $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$.

**Proof.** Due to Lemma 2.5, $D, f_g$, and $g_f$ are uniquely defined. Thus, the two pushout properties have to be shown, i.e. (1) $f_g\circ g=g_f\circ f$ and (2) for each pair of morphisms

[16] The construction provides that $g^{-1}(C_{f_g})=f\triangledown g=f^{-1}(B_{g_f})$.

[17] Here, + denotes the coproduct operator for arbitrary graph structures: if $\mathrm{Sig}=(\mathrm{S},\mathrm{OP})$ is a graph structure and $A$ and $B$ are Sig-algebras, $(A+B)_s=A_s\dot\cup B_s$ for all $s\in\mathrm{S}$ and for all $\mathrm{op}: s\to s'\in\mathrm{OP}$, $\mathrm{op}^{A+B}(x)=\mathrm{op}^A(x)$ if $x\in A_{\mathrm{S}}$ and $\mathrm{op}^{A+B}(x)=\mathrm{op}^B(x)$ if $x\in B_{\mathrm{S}}$. The operator $/\sim$ constructs the quotient of its argument w.r.t. the least congruence which contains the family of relations $\sim=(\sim_s)_{s\in\mathrm{S}}$.

$f': C \to E$ and $g': B \to E$ in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$ such that $f' \circ g = g' \circ f$, there is a unique morphism $u: D \to E$ with $u \circ g_f = g'$ and $u \circ f_g = f'$.

Due to Construction 2.6, the scope of $f_g \circ g$ is $f \bigtriangledown g$ which is also the scope of $g_f \circ f$, and by the identification of $f(z)$ and $g(z)$ for each $z \in f \bigtriangledown g$ in the third and fourth step of the construction, $f_g \circ g = g_f \circ f$. Hence, (1) holds.

In order to prove (2), suppose that there exist $f': C \to E$ and $g': B \to E$ satisfying $f' \circ g = g' \circ f$. Then $B_{g'}$ must be a subalgebra of $B$ whose carriers are contained in $(B - f(A)) \cup f(f \bigtriangledown g)$ and $C_{f'}$ must be a subalgebra of $C$ whose carriers are contained in $(C - g(A)) \cup g(f \bigtriangledown g)$. Since $B_{g_f}$ and $C_{f_g}$ are the largest of those algebras, $B_{g'} \subseteq B_{g_f}$ and $C_{f'} \subseteq C_{f_g}$. With the third and fourth step of the construction, $f_g(x) = f_g(y)$ implies that there is a sequence $z_1, \ldots, z_n \in f \bigtriangledown g$ with $n = 2m+1$ for some $m \in \mathbb{N}$ such that $g(z_1) = x$, $g(z_n) = y$, $f(z_{2i-1}) = f(z_{2i})$, and $g(z_{2i}) = g(z_{2i+1})$ for $i = 1, \ldots, m$. Thus, if $x \in C_{f'}$, $g(z_i) \in C_{f'}$ for $i = 1, \ldots, n$ since $f' \circ g = g' \circ f$. Hence, $y \in C_{f'}$ and $f'(x) = f'(y)$. Similarly, $g_f(x) = g_f(y)$ and $x \in B_{g'}$ implies $y \in B_{g'}$ and $g'(x) = g'(y)$.

With these preliminaries, define $u: D \to E$ by

$$u(x) = \begin{cases} g'(y) & \text{if } x = g_f(y) \text{ and } y \in B_{g'}, \\ f'(y) & \text{if } x = f_g(y) \text{ and } y \in C_{f'}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The morphism $u$ is well-defined since $x = g_f(y_1)$, $x = g_f(y_2)$ and $y_1 \in B_{g'}$ implies $y_2 \in B_{g'}$ and $g'(y_1) = g'(y_2)$ by the remarks above. Similarly, $x = f_g(y_1)$, $x = f_g(y_2)$, and $y_1 \in C_{f'}$ implies $y_2 \in C_{f'}$ and $f'(y_1) = f'(y_2)$. Furthermore, $x = g_f(y)$, $y \in B_{g'}$, $x = f_g(z)$, and $z \in C_{f'}$ implies that there exists $a \in f \bigtriangledown g$ such that $f(a) = y$ and $f_g \circ g(a) = x$. Hence, by $g' \circ f = f' \circ g$, $g'(y) = f'(z)$. Since $B_{g'}$ is closed w.r.t. the equivalence induced by $g_f$ on $B$ and, vice versa, $C_{f'}$ is closed w.r.t. the equivalence induced by $f_g$ on $C$, $u \circ g_f = g'$ and $u \circ f_g = f'$ by definition of $u$. Uniqueness of $u$ follows from the observation that each morphism $v: D \to E$ with $v \circ g_f = g'$ and $v \circ f_g = f'$ requires the same definition on objects as $u$. □

Construction 2.6 has some properties which are used intensively in the following sections.

**Corollary 2.8** (Pushout properties). *If $(D, g_f: B \to D, f_g: C \to D)$ is the pushout object of $f: A \to B$ and $g: A \to C$ in some category* $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$ *of graph structures,*

(1) *$f_g$ and $g_f$ are jointly surjective.*[18]

(2) $\ker(f_g) \subseteq g(\ker(f))$ *and* $\ker(g_f) \subseteq f(\ker(g))$.[19]

(3) *$f_g(g_f)$ is injective if $f(g)$ is injective.*[20]

[18] A partial homomorphism $f: A \to B$ is surjective if $f(A) = B$. Two homomorphisms $f: A \to B$ and $g: C \to B$ are jointly surjective if $f(A) \cup g(C) = B$.

[19] For a partial homomorphism $f: A \to B$, the kernel of $f$ is a subset of its scope defined by $\ker(f) = \{x \in A_f \mid \text{there exists } y \in A_f \text{ such that } x \neq y \text{ and } f(y) = f(x)\}$.

[20] A partial homomorphism $f: A \to B$ is injective if $f(x) = f(y)$ implies $x = y$ for all $x, y \in A_f$.

(4) *$f_g$ and $g_f$ are total if $g$ and $f$ are total. (Thus, each pushout in the category of total homomorphisms is also a pushout in the category of partial homomorphisms.)*

(5) *$g_f$ is total if and only if* (1) $A_f \subseteq A_g$ *and* (2) $g(x)=g(y)$ *implies either* $x, y \in A_f$ *or* $x, y \notin A_f$.

**Proof.** (1) and (2) are direct consequences of Construction 2.6. (3) is implied by (2). If $f$ and $g$ are total $A_f = A = A_g = f \triangledown g$ and, therefore, $B_{g_f} = B$ and $C_{f_g} = C$ which implies (4). In (5), $A_f \subseteq A_g$ and $g(x)=g(y) \Rightarrow x, y \in A_f$ or $x, y \notin A_f$ implies $f \triangledown g = A_f$, which immediately provides $B_{g_f} = B$. Conversely, if $A_f \nsubseteq A_g$, $B_{g_f} \neq B$ and $g_f$ is partial. Also, if there exist $x, y$ with $g(x)=g(y)$, $x \in A_f$, and $y \notin A_f$, we obtain $f \triangledown g \neq A_f$. This implies, by Construction 2.6, that $B_{g_f} \neq B$. Hence, $g_f$ is partial. □

The existence of pushouts in $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ for each graph structure Sig guarantees that $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ is complete w.r.t. arbitrary finite colimits.

**Proposition 2.9** (Initial and final graph structure). *If* Sig *is a graph structure,* $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ *has an initial and final object.*

**Proof.** Let $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ and define $\emptyset_{\mathsf{Sig}}$ by $\emptyset_{\mathsf{Sig},s}=\emptyset$ for all $s \in \mathsf{S}$ and $\mathsf{op}^{\emptyset_{\mathsf{Sig}}}=\emptyset$ for all $\mathsf{op} \in \mathsf{OP}$. The so-defined empty Sig-algebra is both initial and final in $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. For initiality, we need a unique partial homomorphism $f: \emptyset_{\mathsf{Sig}} \to A$ for each $A \in \mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. There is exactly one, i.e. $f=\emptyset$. Conversely, there is exactly one partial homomorphism, namely $\emptyset: A \to \emptyset_{\mathsf{Sig}}$, for each $A \in \mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. □

**Corollary 2.10** (Co-completeness). $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ *is finitely co-complete if and only if* Sig *is a graph structure.*

**Proof.** Direct consequence of Propositions 2.3 and 2.9, Theorem 2.7, and the fact that categories which have all pushouts and an initial object are finitely co-complete; cf. [21]. □

## 3. Single-pushout transformations

This section introduces the basic notions for single-pushout transformations on arbitrary graph structures. We first show (in Section 3.1) that all graph-like structures like graphs, labeled graphs, and hypergraphs and many more complex objects can be seen as algebras w.r.t. a suitable graph structure. Section 3.2 introduces the fundamental notions rule, redex, direct transformation, transformation, and language. Section 3.3 is dedicated to the comparison of single- and double-pushout transformations on labeled graphs. It turns out that single-pushout transformations generalize the classical framework since no application condition is required for redices of single-pushout rules. The effects which rule application at these unrestricted redices can produce are investigated by a small database example in Section 3.4.

### 3.1. *Sample graph structures*

Graph structures are special signatures with the property that the associated category of algebras and partial homomorphisms is finitely co-complete (cf. Section 2). For single-pushout constructions in these categories to provide a reasonable transformation concept, it is to show that objects like graphs or hypergraphs can be seen as graph structures. This is done by presenting the suitable signatures.

**Example 3.1** (*Unlabeled graphs*). Unlabeled graphs consist of a set of vertices $V$ and a set of edges $E$. Each edge is connected to its source and target vertex by a *monadic* operation. Hence, the associated graph structure is:

Unlabeled Graphs =
**Sorts** V, E
**Operations**
source, target : $E \to V$

**Example 3.2** (*Edge-labeled graphs*). If the edges of a graph are labeled by elements of a label set $L$, we obtain a natural decomposition of the edge set into sets of edges with the same label. Hence, the edge set of edge-labeled graphs is an $L$-indexed family:

Edge-Labeled Graphs =
**Sorts** V, $(E_l)_{l \in L}$
**Operations**
(source, target : $E_l \to V)_{l \in L}$

The family of edges $(E_l)_{l \in L}$ and the corresponding family of operators can be infinite if $L$ is. The theory of Section 2, however, is also applicable to these infinite structures since all operators are monadic.

Note that the Edge-Labeled-Graph-homomorphisms are label-preserving.

**Example 3.3** (*Labeled graphs*). Labeled graphs are constructed from edge-labeled graphs by sorting the vertices w.r.t. their labels taken from a vertex label set $M$:

Labeled Graphs =
**Sorts** $(V_m)_{m \in M}$, $(E_{sm, tm, l})_{sm, tm \in M, l \in L}$
**Operations**

$$\begin{pmatrix} \text{source} : E_{sm, tm, l} \to V_{sm} \\ \text{target} : E_{sm, tm, l} \to V_{tm} \end{pmatrix}_{sm, tm \in M, l \in L}$$

The structure of the operator symbols must be so complex since the associated homomorphisms shall preserve the labels of the graph elements. Hence, every edge is not only distinguished by its own label but also by the labels of its source and target vertex.

**Example 3.4** (*Unlabeled hypergraphs*). Hypergraphs allow their edges to be connected to more than one source and more than one target. Therefore, the set of

hyperedges $H=(H_{n,m})_{n,m\in\mathbb{N}}$[21], is a family of edge sets and each $h\in H_{n,m}$ has $n$ sources and $m$ targets.

Unlabeled Hypergraphs=

**Sorts** $V, (H_{n,m})_{n,m\in\mathbb{N}}$

**Operations**

$(\text{source}_1, \ldots, \text{source}_n, \text{target}_1, \ldots, \text{target}_m : H_{n,m} \to V)_{n,m\in\mathbb{N}}$

Note that the Unlabeled-Hypergraph-homomorphisms must respect the type of the edges, i.e. edges can only be mapped to edges with the same number of source and target connection.

Labeled hypergraphs can be obtained from hypergraphs in the same way we have constructed labeled graphs from unlabeled graphs.

If the distinction between source and target connections is dropped, we obtain *undirected* hypergraphs. If more than two different connection types are used, multi-dimensional objects as they are applied, for example, in [38] can be represented.

**Example 3.5** (*Signatures*) Parisi-Presicce [34] applies graph transformation techniques to specify signature manipulations. The aim is to provide a method for rule-based software design. If signatures, i.e. $\text{Sig}=(S, OP=(OP_{w,s})_{w\in S^*, s\in S})$, are considered as a special type of hypergraphs (see below), single-pushout transformations can also be applied to these structures.

Signature=

**Sorts** Sorts, $(\text{Operators}_n)_{n\in\mathbb{N}}$

**Operations**

$(\text{arg}_1, \ldots, \text{arg}_n, \text{value} : \text{Operators}_n \to \text{Sorts})_{n\in\mathbb{N}}$

**Example 3.6** (*Functional expressions*). Functional expressions over a signature Sig are hyperpaths w.r.t Sig. Sets of these hyperpaths can also be modeled as graph structures. The graph structure Signature above has to be slightly changed: substitute for each sort symbol a set of instances of the sort and for each operator symbol a set of instances of the operator. The signature Sig prescribes which sort instances are allowed as arguments or values for an operator instance. This relation is expressed by the graph structure Expressions(Sig) below which can be defined for each signature $\text{Sig}=(S, OP)$:

Expressions(Sig=(S, OP))=

**Sorts** $(\text{Sort Instances}_s)_{s\in S}, (\text{Operator Instances}_{op})_{op\in OP}$

**Operations**

$$\left(\begin{array}{l} \text{argument}_1 : \text{Operator Instances}_{op} \to \text{Sort Instances}_{s_1} \\ \quad \vdots \\ \text{argument}_n : \text{Operator Instances}_{op} \to \text{Sort Instances}_{s_n} \\ \text{value} : \text{Operator Instances}_{op} \to \text{Sort Instances}_{s} \end{array}\right)_{(op: s_1, \ldots, s_n \to s)\in OP}$$

[21] $\mathbb{N}$ denotes the set of natural numbers with zero.

Jungles as they are used in [35, 20, 22] are special expressions. They do not admit cyclic structures and sort instances which are value of two different operator instances. Each jungle can be interpreted as a set of finite Sig-terms with variables: The variables are exactly the sort instances which are not value of any operator instance in the jungle. If we interpret the value connection of operator instances as the source of a hyperedge and the argument connections as targets, each sort instance $s_i$ in a jungle represents the term which corresponds to the hyperpath from $s_i$ to variables. The term interpretation of a jungle is the set of these terms. Note that due to different degree of "sharing" for common subterms, different jungles (and expressions) can represent the same set of terms.

The same interpretation leads to infinite terms for cyclic expressions. And the situation that a sort instance $s_i$ is value of two different operator instances can be interpreted as an equation: Take all hyperpaths from $s_i$ to variables and interpret them as possibly infinite terms. The set of equations encoded in the expression at $s_i$ consists of all pairs of these terms. The set of equations encoded in an expression is the union of the equations which are encoded at the sort instances of the expression. Hence, the interpretation of jungles as sets of terms corresponds to the interpretation of expressions as sets of equations, i.e. the jungle interpretation is a special case of the expression interpretation. With these ideas, each expression w.r.t. a signature Sig is an equational specification w.r.t. Sig (cf. [34]).

Example 3.7 demonstrates that graph structures are flexible enough to represent very complex objects:

**Example 3.7** (*Structure of graph transformation implementations*). The implementation of algebraic graph transformation currently being developed at the Technical University of Berlin uses so-called ALR-graphs as the fundamental data structure [2]. ALR-graphs not only allow to represent arbitrary labeled graphs but also morphisms between graphs. Since morphisms map vertices to vertices and edges to edges, they are represented by pairs of *vertex assignments* and *edge assignments*.[22] In order to keep track of which assignment belongs to which morphism, an abstraction operator is introduced in ALR-graphs which allows to group vertices and edges into graphs and vertex and edge assignments into morphisms. Thus, ALR-graphs as algebras w.r.t. the graph structure below are able to represent the diagram level (graphs and morphisms) and the object level (vertices, edges, and assignments) in a single structure.

ALR-Graph =
  **Sorts** V, E, V-Ass, E-Ass, Graph, Morphism
  **Operations**
    s, t : E → V
    s, t : V-Ass → V

[22] Note that edge assignments are objects on a third level if we think of vertices being primary objects and edges being secondary items.

s, t: E-Ass→E
s, t: Morphism→Graph
abstract: V→Graph
abstract: E→Graph
abstract: V-Ass→Morphism
abstract: E-Ass→Morphism

In the implementation of ALR-graphs, context conditions make sure that the abstraction relation and the morphisms satisfy the intuitive requirements, for example:

(1) for all $e \in \mathsf{E}$, $\mathsf{abstract}(e) = \mathsf{abstract}(\mathsf{s}(e)) = \mathsf{abstract}(\mathsf{t}(e))$,

(2) for each $e \in \mathsf{E\text{-}Ass}$, there exist $v, w \in \mathsf{V\text{-}Ass}$ such that $\mathsf{abstract}(e) = \mathsf{abstract}(v) = \mathsf{abstract}(w)$ and $\mathsf{s}(\mathsf{s}(e)) = \mathsf{s}(v)$, $\mathsf{s}(\mathsf{t}(e)) = \mathsf{t}(v)$, $\mathsf{t}(\mathsf{s}(e)) = \mathsf{s}(w)$, and $\mathsf{t}(\mathsf{t}(e)) = \mathsf{t}(w)$,

(3) and some more; cf. [2].

Although these conditions are equations in most cases, the graph transformation approach with partial morphisms cannot be adapted to the full subcategory of all ALR-graphs which satisfy the requirements. This is due to the fact that every nontrivial generated congruence[23] on objects cannot be extended to a free construction in the context of partial morphisms. Thus, the intuitive consistence requirements above can only be used as correctness criteria for transformations performed in $\mathsf{Alg}^{\mathsf{P}}(\mathsf{ALR\text{-}Graph})$.

Application of graph transformation rules in such a system means building of some pushout squares of appropriate morphisms. This is due to the fact that the data structure of ALR-graphs allows to represent all features of algebraic graph transformation, i.e. graphs, morphisms, and redices.

On the other hand, ALR-graphs are graph structures themselves. Thus, the implementation of graph transformation on the basis of ALR-graphs can be seen as a graph transformation system manipulating graph transformation systems.

### *3.2. Basic notions*

Section 3.1 has presented a variety of graph-like structures as *graph structures*. Hence, it is worthwhile to formulate the single-pushout transformation concept for arbitrary graph structures.

**General assumption 3.8.** In the following definitions and propositions, it is assumed that all objects and homomorphisms are taken from a fixed category $\mathsf{Alg}^{\mathsf{P}}(\mathsf{Sig})$ for some graph structure Sig.

The mathematical basis for rules, redices, and their interaction within a direct transformation is provided by the pushout construction for partial homomorphisms in Construction 2.6.

[23] The generated congruence is not trivial, if it differs from $\Delta_G$ (the least reflexive relation) for at least one object $G$.

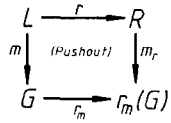


Diagram 6. Direct transformation in the single-pushout approach.

**Definition 3.9** (*Rules, redices, and direct transformation*). A *transformation rule* $r: L \to R$ is a partial morphism from the *left-hand side* of the rule $L$ to the *right-hand side* $R$. A *redex* for $r$ in some object $G$ is a total morphism $m: L \to G$ from the left-hand side of the rule to $G$. The application of a rule $r: L \to R$ to an object $G$ at a redex $m: L \to G$ *transforms* $G$ to $r_m(G)$ which is the pushout object in Diagram 6.

Note that the graph $G$ and the direct derivation $r_m(G)$ are connected by the pushout morphism $r_m: G \to r_m(G)$ which is also called *direct transformation morphism* below. We distinguish the following types of redices.

**Definition 3.10** (*Application conditions*). Let $r: L \to R$ be a transformation rule and $m: L \to G$ a redex for $r$ in $G$.

(1) The redex $m$ is *conflict-free* if $m(x) = m(y)$ implies $x, y \in L_r$ or $x, y \notin L_r$.

(2) If $m(x) = m(y)$ implies $x = y$ or $x, y \in L_r$, $m$ is called *d-injective*.

(3) The redex $m$ is *d-complete* if for each object $o \in G$ with $\mathsf{op}^G(o) \in m(L - L_r)$ for some operator $\mathsf{op} \in \mathsf{Sig}$, we have $o \in m(L - L_r)$.

Redices with these additional features will turn out to impose special properties on direct transformations which make the whole transformation process more transparent. But also from the intuitive point of view, these application conditions are natural. If we reconsider the basic ideas about graph transformation of Section 1 in this framework of graph structures and partial morphisms, we can again single out three components of a rule: the part meant to be deleted, i.e. $L - L_r$, the subobject of $L$ which shall be preserved, i.e. $L_r$, and the added structure $R - r(L)$ (forget about identification of $r$ for the moment).

With these interpretations, conflict-freeness of a redex guarantees that an element of $G$ is *either* meant to be preserved *or* meant to be deleted. The general concept of redices allows conflicts in this respect and the transformation process has to solve the conflict by defining deletion or preservation to be dominant (compare Section 3.4).

The notion of d-injectivity implies conflict-freeness and additionally requires one-to-one correspondence between candidates for deletion in $G$ and $L$. Thus, in order to apply a rule which deletes $n$ items, we have to find $n$ suitable elements in $G$ if d-injective redices are required.

D-complete redices, on the other hand, make sure that the whole *structural context* of the elements of $G$ which are going to be deleted is described in $L$. Here $x$ is in the structural context of $y$ if $\mathsf{op}(x) = y$ for some operator symbol $\mathsf{op}$ in the underlying graph structure. For example, the structural context of a vertex is given by all incident edges in the category of directed graphs.

These properties of d-injective or d-complete redices can be summarized as follows: if $L - L_r$, i.e. the part of the rule's left-hand side which describes the elements that are deleted by rule application, has $n$ elements, d-injectivity of redices guarantees that at least $n$ items are deleted in each direct transformation and d-completeness makes sure that at most $n$ elements are deleted.[24]

In Definition 3.9 of direct transformation, this intuition is exactly captured as the following propositions show.

**Proposition 3.11** (Direct transformation). *Let a direct transformation $r_m: G \to H$ be given as it is defined by Definition* 3.9.

(1) *If $m$ is conflict-free, then $r \triangledown m = L_r$, the embedding of the rule's right-hand side in the transformation result $m_r: R \to H$ is total, $m(L_r) \subseteq G_{r_m}$, and $m(L - L_r) \subseteq G - G_{r_m}$.*

(2) *If $m$ is d-injective and d-complete, $G - G_{r_m} = m(L - L_r)$.*

**Proof.** Direct consequence of the pushout Construction 2.6 and Corollary 2.8. □

On the basis of the notion for direct transformations, we can give precise meaning to the notions "rule system", "transformation", and "generated language".

**Definition 3.12** (*Rule system, transformation, language*). A *rule system* $RS$ is a finite set of transformation rules.

An object $G$ can be *transformed* to $H$ with a rule system $RS$ if there is a sequence of direct transformations $(r^i)_{m^i}: G^{i-1} \to G^i$ for $i = 1, \ldots, n$ such that $G = G^0$, $H = G^n$, and for $i = 1, \ldots, n$, $(r^i: L^i \to R^i) \in RS$ and $m^i: L^i \to G^{i-1}$ is a redex for $r^i$ in $G^{i-1}$.

The *language* generated by a rule system $RS$ with *start object* $G$ is denoted by $RS(G)$ and defined by $RS(G) = \{H \mid G \text{ can be transformed to } H \text{ with } RS\}$.

$RS_{cf}(G)$, $RS_i(G)$, and $RS_{i+c}(G)$ denote the sublanguages of $RS(G)$ which are generated by $RS$ using conflict-free, d-injective, respectively d-injective and d-complete redices in each direct transformation only.

If $G$ transforms to $H$ with rules in $RS$, $G$ and $H$ are connected by the partial morphism $R_M = r^n_{m^n} \circ \cdots \circ r^1_{m^1}: G \to H$ which is called transformation morphism in the following.

### *3.3. Single- versus double-pushout transformations*

For the comparison of single- and double-pushout transformations assume that all constructions in this paragraph are performed in the category of Labeled Graphs-algebras; cf. Example 3.3. See Appendix A for basic notions of the double-pushout approach.

**Definition 3.13** (*Translation of single- and double-pushout rules*). If $r: L \to R$ is a transformation rule according to Definition 3.9, $D(r) = (l: L_r \to L,\ r': L_r \to R)$ denotes its

[24] The application conditions d-injectivity and d-completeness reformulate the gluing conditions of the double-pushout framework for single-pushout transformations.

translation to a double-pushout rule, where $l$ is the inclusion of $L_r$ in $L$ and $r'$ is the domain restriction of $r$ to $L_r$.

Conversely, for a double-pushout rule $p=(l: K \to L, r: K \to R)$, $S(p): L \to R$ denotes its translation to single-pushout rules, where $L_{S(p)}=l(K)$ and $S(p)=r \circ l^{-1}$.[25]

**Theorem 3.14** (Embedding of the classical approach). *If the object $H$ is the result of transforming an object $G$ with rule $p$ at redex $m$ in the dobule-pushout framework, the translation of $p$ to a single-pushout rule, i.e. $S(p)$, transforms $G$ to $H$ at the same redex $m$ in the single-pushout setting.*

*Conversely, if $G$ can be transformed to $H$ with rule $r$ at redex $m$ by a single-pushout transformation, the translation of $r$ to a double-pushout rule, i.e. $D(r)$, is applicable to $G$ at $m$ in the double-pushout framework if and only if $m$ is d-injective and d-complete. In this case, the double-pushout transformation of $G$ with $D(r)$ at $m$ results in the same object $H$.*

**Proof.** For the first part, consider Diagram 7, where (1)+(2) depicts a direct transformation in the double-pushout setting and $s$ and $s^*$ are the translations of $(l, r)$ and of $(l^*, r^*)$ to single-pushout rules, i.e. $s=S(l, r)$ and $s^*=S(l^*, r^*)$. We have to show that (3) is a pushout in the framework of partial morphisms. By Theorem A.5, $m$ satisfies the gluing conditions. Thus, it is d-injective and d-complete w.r.t. $s$. Thereby, it is conflict-free providing $s \triangledown m=L_s=l(K)$ by Proposition 3.11. Furthermore, the pushout morphisms $s_m$ and $m_s$ satisfy $R_{m_s}=R$ and $G_{s_m}=G-m(L-L_s)=D$ by the same proposition. Therefore, $s_{|(s \triangledown m)}=r$ and $m_{|(s \triangledown m)}=k$. Since (2) is the pushout of $r$ and $k$, $H$ is the pushout of $s_{|s \triangledown m}$ and $m_{|s \triangledown m}$ and thereby coincides with the pushout object of $s$ and $m$ in the framework of partial homomorphisms; cf. Construction 2.6 and Diagram 5. Since $l^*: D \to G$ is the inclusion of $G_{s^*}=D$ into $G$ and $R_{m^*}=R$, $s^*$ and $m^*$ are the pushout morphisms for $s$ and $m$.

For the second part, consider Diagram 8, where (1) is a direct single-pushout transformation and $(l, r)$ and $(l^*, r^*)$ are the translations of $s$ and $s_m$ to double-pushout

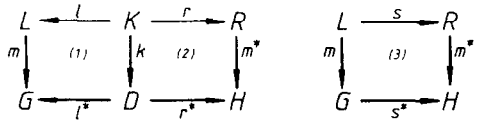


Diagram 7. Translation of double-pushout diagrams.

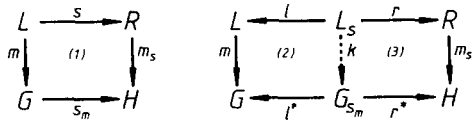


Diagram 8. Transformation of single-pushout diagrams.

[25] Note that $S(p)$ is well-defined since $l$ is supposed to be injective in the double pushout setting.

rules, respectively, i.e. $(l, r)=D(s)$ and $(l^*, r^*)=D(s_m)$. The rule $(l, r)$ is applicable to $G$ at $m$ if and only if $m$ satisfies the gluing conditions *identification 2* and *dangling* of Theorem A.5. These conditions are satisfied if and only if $m$ is d-injective and d-complete w.r.t. $s$.

If $m$ is d-injective and d-complete, Proposition 3.11 provides $m(L_s) \subseteq G_{s_m}$. Thus, we can define $k=(l^*)^{-1} \circ m \circ l$ as a *total* homomorphism. Since the so-defined morphism satisfies $k=m_{|L_s}=m_{|S \triangledown m}$ and we have $r=s_{|L_s}=s_{|S \triangledown m}$ by Definition 3.13 and Proposition 3.11, (3) is a pushout diagram of total homomorphisms by Construction 2.6; compare also Diagram 5. Square (2) commutes by definition of $k$, $l$ and $l^*$ are injective, and $m$ is injective outside of $l(L_s)$. This implies that (2) is a pushout of graph structures as well; cf. Construction 2.6. □

Theorem 3.14 shows that each transformation of graphs in a double-pushout framework corresponds to a single-pushout transformation with the translated rule. Vice versa, the whole theory for double-pushout transformations can be reobtained by restricting the single-pushout approach to d-injective and d-complete redices.

### *3.4. Example: a small police database system*

The power of the new concept lies in its ability to perform transformations even if the redices are not d-complete and d-injective. Thus, the single-pushout approach is free from any other precondition for rule application than finding a homomorphic image of the rule's left-hand side in the actual object that shall be manipulated.

The following small police database example demonstrates the usefulness of this property. It has been inspired by the information processing system of (W-) Germany's police INPOL [31]. This database mainly consists of two types of data, namely personal data and case data. Therefore, the initial state (i.e. the empty database) is characterized by the number of personal and case databases in the system. Having just one of each sort, we obtain the graph in Fig. 1 as initial state.[26] The following operations manipulate the database states:

(1) Add person $p$ to the personal database.

(2) Open a new case $c$ in the database for cases.

(3) Relate person $p$ in kind $k$ to a case $c$. (The kind can be $s$ for suspected person, $w$ for witness, $v$ for victim, etc.)


Fig. 1. Initial state.

[26] The whole example is based on the graph structure Labeled Graphs of Example 3.3. The type of the vertices, i.e. black/white or big/small, must be interpreted as part of the label.

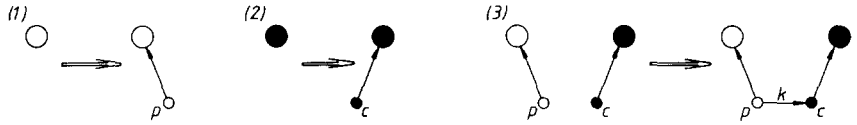


Fig. 2. Object and relation creation.

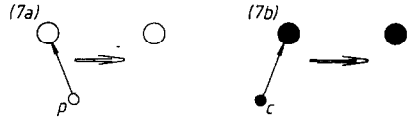


Fig. 3. Object deletion.

(4) Relate person $p$ in kind $k$ to another person $q$. (Kinds are, for example, $f$ for father of, $b$ for brother, etc.)

(5) Relate case $c$ in kind $k$ to another case $d$. ($s$ for subcase, etc.)

(6) Erase a relation. (For example: drop suspect against $p$ in $c$.)

(7) Erase database entries. (That is, erase data concerned with person $p$, close case $c$, etc.)

(8) More complex operations which combine several basic functions in a single step.

The graph transformation model for the operations of type 1–3 is given by the rules in Fig. 2.[27]

Operations of type 4 and 5 have the same scheme as the rule (3) in Fig. 2 but they work on personal or case data exclusively. The erasure operations of type 6 and 7 are modeled by the corresponding inverse rules of type 1–5. Inverse rules can be constructed as long as the rule morphism is injective since the inverse of an injective partial morphism is itself an (injective) partial morphism. Figure 3 visualizes the rules for database entry deletion. More complex operations (type 8) can be built from the basic ones (type 1–7) using sequential composition, parallel composition, and amalgamation formally investigated in Sections 4, 5, and 6, respectively. The rule in Fig. 4, for example, is a parallel composition constructed from the rules "erase person $p$" and "relate person $q$ in kind "father of" to person $r$".

Figures 5 and 7 show some direct transformations with these rules. Figure 5 demonstrates that single-pushout transformations are able to express "deletion in unknown contexts". Due to Construction 2.6(2), the erasure of the $q$-labeled vertex (representing a person in the database) by the corresponding "person data deletion rule" triggers the

[27] Partial morphisms are drawn as double arrows. The mapping of the objects is indicated by the graphical arrangement: The morphism maps all objects of its domain which occur at the same relative position in the codomain. This works as long as the morphisms are injective. Noninjective morphisms will be indicated by corresponding natural numbers which are used as object identifiers in these cases.

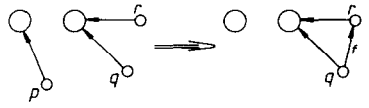

Fig. 4. Derived rule.

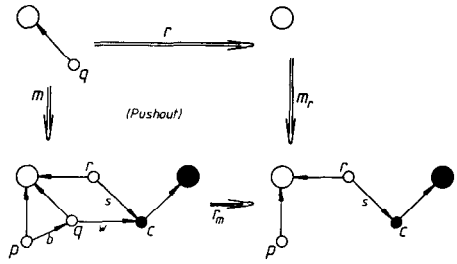


Fig. 5. Deletion in unknown contexts.

Fig. 6. Database reset.

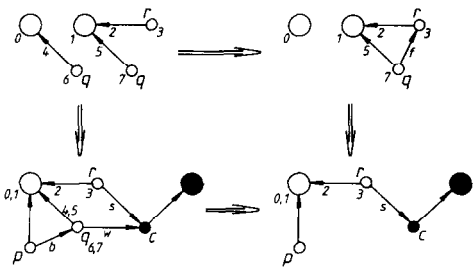

Fig. 7. Noninjective redices.

erasure of all incident edges of this vertex from the domain of the transformation morphism $r_m$.[28]

This operational behavior of the transformation process enables to describe a complete reset of a personal database in the system by the rule which is given as the empty morphism in Fig. 6. Its application erases all connections of the persons in the database to the database root. Hence, no rules for these persons are applicable afterwards.

Figure 7 visualizes a rule application at a redex which is not conflict-free. The parallel rule of Fig. 4 is used with both subactions manipulating data concerned with person $q$: delete $q$'s data and insert the information that $q$ is father of $r$. As it is described in Construction 2.6(1), deletion is dominant w.r.t. preservation.[29] Due to

[28] Note that the double-pushout translation of this rule is not applicable in the situation of Fig. 5 due to a violation of the dangling condition (cf. Section 3.3). Hence, complete person data deletion in our example is not directly expressible in the double-pushout framework. But it seems to be mere accident that exactly this operation is most problematic in the real INPOL System of the German police. First of all, the police tried to prevent this operation from being implemented at all since they always fear that deletion of data can make "their knowledge of the world" incomplete; a conception they simply hate. Secondly, after they were forced to implement it by data protection laws, they persistently refused to apply it or managed to produce a new copy before the actual deletion. This behavior and the redundant architecture of the system led to a data structure that, thirdly, prohibits any complete deletion of *all* data concerned with a single person even if the official in charge actually wants to erase it (compare [31] for a detailed discussion).

[29] From the data protection point of view, it is the way it should be in this example.

vertex 7 being in the scope of the rule and vertex 6 outside, the identification of these vertices by the redex forces vertex (6, 7) to be outside the scope of the transformation morphism. A side effect is that vertex 7 of the rule's right-hand side cannot be mapped to the transformation result by the corresponding pushout morphism. Hence, the embedding of the right-hand side into the transformation result is partial for conflicting redices.

## 4. Sequential composition

The easiest way to construct new rules from a given rule system $RS$ is to consider direct transformations $r_m: G \to H$ as rules themselves, so-called *rule-derived rules*. Since rules are only required to be (partial) morphisms, direct transformations possess the right structure.

Within the single-pushout approach, we can even do more: If there is a transformation of $G$ to $H$ according to Definition 3.12 by a sequence of rules $R = r^1, \ldots, r^n$ at a sequence of redices $M = m^1, \ldots, m^n$, $G$ and $H$ are again connected by a partial morphism, i.e. the transformation morphism $R_M$. Thus, all transformations in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$ have the same structure, the structure of a transformation rule. This allows to interpret all transformations with a rule system $RS$ as *derived rules*.

**Definition 4.1** (*Rule-derived rule*). A rule $rd: G \to H$ is a *rule-derived rule* w.r.t. a rule system $RS$ if there is a rule $r \in RS$ and a redex $m$ for $r$ in $G$ such that $rd$ coincides with the direct transformation $r_m: G \to r_m(G)$, i.e. $H = r_m(G)$ and $rd = r_m$. The *closure w.r.t. rule-derived rules* $RS^{\mathrm{D}}$ is the least rule system which satisfies (1) $RS \subseteq RS^{\mathrm{D}}$ and (2) if $r$ is rule-derived from $RS^{\mathrm{D}}$, $r \in RS^{\mathrm{D}}$.

**Theorem 4.2** (Rule-derived rule). *If $K$ is directly transformed to $M$ with a rule-derived rule $rd$, there is a direct transformation of $K$ to $M$ with the original rule from which $rd$ is derived.*

**Proof.** Consider Diagram 9. The existence of a direct transformation from $K$ to $M$ with the rule-derived rule $rd$ implies that there is a redex $n$ such that (2) is a pushout square. The property of $rd$ being rule-derived ensures that there is a rule $r$ and a redex

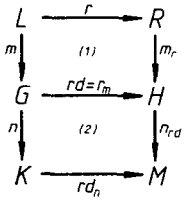


Diagram 9. Application of rule-derived rule.

$m$ for $r$ in $G$ such that (1) is a pushout diagram. Since pushouts compose, (1)+(2) is a pushout. It is the diagram for the application of $r$ at the redex $n \circ m$ which is total because both components are. Thus, $K$ can be transformed to $M$ using $r$ at $n \circ m$ and $rd_n = r_{(n \circ m)}$ due to the uniqueness of pushouts. □

**Corollary 4.3** (Generated language). *The language generated by a rule system RS coincides with the language generated by the closure $RS^D$ for all start objects G, i.e. $RS(G) = RS^D(G)$.*

**Proof.** $RS(G) \subseteq RS^D(G)$ follows directly from $RS \subseteq RS^D$. $RS^D(G) \subseteq RS(G)$ is a direct consequence of Theorem 4.2. □

General derived rules are more complicated.

**Definition 4.4** (*Derived rule*). A rule $rd: G \to H$ is a *derived rule* w.r.t. a rule system $RS$ if $rd = R_M: G \to H$ for a sequence of rules $R = r^1, \ldots, r^n \in RS$ and a sequence of redices $M = m^1, \ldots, m^n$ for these rules. The *closure w.r.t. derived rules* $RS^T$ is the least rule system satisfying (1) $RS \subseteq RS^T$ and (2) if $r$ is derived from $RS^T$, $r \in RS^T$.

Using arbitrary transformations as derived rules, we loose the properties of Corollary 4.3.

**Example 4.5** (*Derived rule*). Consider Section 3.4, especially the rule of Fig. 6. Applying this rule twice to a graph containing two personal databases provides us with the derived rule *rd* in Fig. 8.

The derived rule *rd* can now be applied to a graph with only one person database at a noninjective redex. Thus, a system state containing a single personal database can be transformed to a graph with two of these databases if all derived rules are allowed for transformations. This cannot be done with the original rule system: it is easy to check that all rules preserve the number of vertices representing databases.

**Theorem 4.6** (Derived rule). *If rd is a derived rule w.r.t. a rule system RS, p is a d-injective redex for rd in G, and $rd_p: G \to H$ is the corresponding direct transformation, then G can be transformed to H using the rules in RS only.*

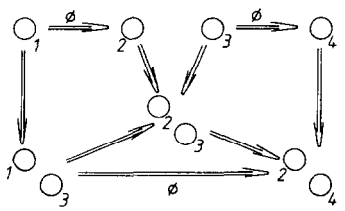


Fig. 8. Example of a derived rule.

**Proof.** If $rd$ represents a sequence of length 0, it is the identity by definition. Applying $rd$ to $G$ in this case results in $G$ which is also the result using the empty sequence of rules in $RS$.

If $rd$ represents a transformation sequence of length 1, the proposition specializes to the case of Theorem 4.2.

Thus, it remains to consider the case that $rd$ represents a transformation sequence whose length is greater or equal 2. If we manage to prove the statement of the theorem for derived rules whose corresponding transformation sequence has exactly length 2, we are done. All other cases follow by a simple induction on the length of the transformation sequence which $rd$ represents.

The situation that $rd$ has been derived from a transformation of length 2 is depicted in Diagram 10. The rules $r$ and $s$ are contained in the rule system $RS$. The derived rule $rd$ is given by $rd = s_n \circ r_m : G^0 \to G^2$. The subdiagrams (1) and (2) are the corresponding direct transformations. The rectangle (3)+(4) represents the direct transformation of $G$ with $rd$ at the redex $p$.

Since $rd = s_n \circ r_m$, we can decompose (3)+(4) into two pushouts (3) and (4). The proof is completed if it can be shown that $w \circ n$ is a redex for $s$. Under this premise (1)+(3) depicts a direct transformation with the rule $r$, (2)+(4) visualizes a direct transformation with the rule $s$, and, therefore, $G$ can be transformed to $H$ using rules in $RS$ only.

The redex $p$ is d-injective w.r.t. $s_n \circ r_m$ by assumption. Since $G_{(s_n \circ r_m)} \subseteq G_{r_m}$, $p$ is conflict-free w.r.t. $r_m$, which provides by Proposition 3.11 that the morphism $w$ is total. Since $n$ is a redex, $w \circ n$ is total and a redex for $s$ in $K$.

Uniqueness of colimits guarantees $rd_p(G) = H = s_{(w \circ n)}(r_{(p \circ m)}(G))$. □

A direct consequence of Theorem 4.6 is that each transformation $r^n_{m^n} \circ \cdots \circ r^1_{m^1} : G \to H$ can be replayed in bigger contexts $K$. That is, if there is an inclusion $i : G \to K$, there is a transformation $r^n_{i^n \circ m^n} \circ \cdots \circ r^1_{i^1 \circ m^1} : K \to M$ such that $i^1 = i$ and for $j = 1, \ldots, n$, $i^j$ is an inclusion. The final result $M$ of this replay is given by applying the corresponding derived rule $r^n_{m^n} \circ \cdots \circ r^1_{m^1} : G \to H$ at the redex $i$.

**Corollary 4.7** (Generated language). *Let $RS$ be a rule system, $RS^T$ its closure w.r.t. derived rules, and $G$ an arbitrary start object.*

(1) $RS(G) \subseteq RS^T(G)$.

(2) *For some systems,* $RS(G) \neq RS^T(G)$.

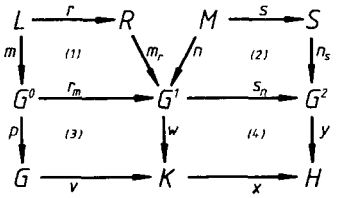


Diagram 10. Direct transformation with derived rule.

(3) *If the redices for the construction of derived rules and the redices for direct transformations with rules and derived rules are restricted to d-injective ones, RS generates the same language as* $RS^T$, *i.e.* $RS_i(G) = RS_i^T(G)$.

**Proof.** (1) is obvious since $RS \subseteq RS^T$. (2) is shown by Example 4.5. $RS_i(G) \subseteq RS_i^T(G)$ in (3) is trivial since $RS \subseteq RS^T$. For the reverse inclusion, we must show that the redices $p \circ m: L \to G$ and $w \circ n$: $M \to K$ constructed in the proof of Theorem 4.6 are d-injective (cf. Diagram 10). By the assumption that redices are restricted to d-injective ones, $m$, $n$, and $p$ in Diagram 10 are d-injective.

Suppose that $x \neq y$ and $p \circ m(x) = p \circ m(y)$. If $m(x) = m(y)$, $x, y \in L_r$ and we are done. If $m(x) \neq m(y)$, $m(x), m(y) \in G^0_{(s_n \circ r_m)} \subseteq G^0_{r_m}$ since $p$ is d-injective, and Proposition 3.11(1) provides $x, y \in L_r$. Thus, $p \circ m$ is d-injective.

Corollary 2.8(2) provides that $x \neq y$ and $w(x) = w(y)$ implies $x, y \in r_m(\ker(p))$. Since $m$ is d-injective, this means that $x, y \in r_m(G^0_{(s_n \circ r_m)})$ and, therefore, $x, y \in G^1_{s_n}$. But this exactly states d-injectivity of $w$ w.r.t. $s_n$. Thus, $n$ is d-injective w.r.t. $s$ and $w$ is d-injective w.r.t. $s_n$ and the same argument given for $m$ and $p$ above provides that $w \circ n$ is d-injective for $s$. □

Among the derived rules of a rule system, a special set of so-called *sequential compositions* can be distinguished which allows to simulate all transformations in the system by appropriate direct transformations.

**Definition 4.8** (*Sequential composition*). The derived rule $s_n \circ r_m$ in Diagram 10 is a *sequential composition* of $r$ and $s$ if $m_r$ and $n$ are jointly surjective.

**Theorem 4.9** (Sequential composition). *For direct transformations* $r_m: G \to H$ *and* $s_n: H \to K$, *there is a sequential composition* $t: N \to T$ *of* $r$ *and* $s$ *and a redex* $i: N \to G$ *such that* $t$ *transforms* $G$ *to* $K$ *at* $i$, *i.e.* $K = t_i(G)$.

**Proof.** Consider Diagram 11. Construct $N = m(L) \cup (r_m)^{-1}(n(M))$. This construction provides a subalgebra of $G$. Let $p = m^{|N}$ be the codomain restriction of $m$ w.r.t. $N$ and $i$ the inclusion of $N$ in $G$. Construct $Y$ as the pushout of $p$ and $r$. Thereby, square (3) is a pushout diagram and $x$ is the unique morphism such that $x \circ p_r = m_r$; it is injective

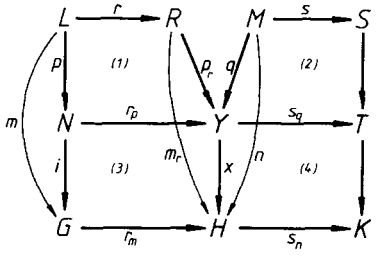


Diagram 11. Short cut by sequential composition.

since $i$ is [cf. Corollary 2.8(3)] and it is total since $i$ is conflict-free [cf. Proposition 3.11(1)].

Since $H$ is the pushout of $r$ and $m$, $r_m$ and $m_r$ are jointly surjective such that $n(M) \subseteq r_m(G) \cup m_r(R)$. By definition of $N$, we get $n(M) \subseteq r_m \circ i(N) \cup m_r(R)$. The square (3) is commutative and $m_r = x \circ p_r$ which implies $n(M) \subseteq x \circ r_p(N) \cup x \circ p_r(R) = x(r_p(N) \cup p_r(R)) = x(Y)$. Hence, $n$ factors through $Y$, i.e. there is a morphism $q$ such that $x \circ q = n$. Construct square (2) as the pushout of $q$ and $s$ which turns subdiagram (4) into a pushout as well.

The last thing to be shown is that $p_r$ and $q$ are jointly surjective. Since $Y$ is the pushout of $r$ and $p$, $p_r$ and $r_p$ are jointly surjective, i.e. $Y = p_r(R) \cup r_p(N)$. Thus, it is to be shown that $r_p(N) \subseteq p_r(R) \cup q(M)$. We know that $x \circ r_p(N) = r_m \circ i(N)$ and by definition of $N$, $r_m \circ i(N) = r_m(m(L) \cup (r_m)^{-1}(n(M))) \subseteq r_m \circ m(L) \cup n(M) = x \circ p_r \circ r(L) \cup x \circ q(M) \subseteq x(p_r(R) \cup q(M))$. Hence, $x \circ r_p(N) \subseteq x(p_r(R) \cup q(M))$ which implies $r_p(N) \subseteq p_r(R) \cup q(M)$ since $x$ is total and injective.

Now take $t = s_q \circ r_p$, which is a sequential composition of $r$ and $s$. The diagram (3)+(4) depicts the direct transformation of $G$ to $K$ with $t$ at $i$ as desired. □

In the general case, there are many compositions of rules; in fact, there are several different compositions even if we fix the jointly surjective pair of morphisms $m_r$ and $n$ (cf. Definition 4.8). Nevertheless, the set of compositions for $r$ and $s$ is always finite if $r$ and $s$ are finite. But it depends on the actual transformation situation which one is to choose in order to simulate a concrete transformation sequence.

**Corollary 4.10** (Abstracting from transformations). *For a rule system RS which is closed under sequential composition, every transformation in RS coincides with a* direct *transformation in RS.*

**Proof.** Direct consequence of Theorem 4.9. □

## 5. Parallel Composition

Parallel composition of rules provides a model for simultaneous application of two or more rules. The simultaneous application is represented by the application of the *parallel rule* which is given by the disjoint union of some rules. The main question is: can the effect of parallel rule transformations be simulated by sequential transformations with the components of the parallel rule? The answer in the classical framework is an unrestricted "yes" [8]. We show that the answer is positive in the new approach only if redices are restricted to d-injective ones. Parallel rule application at arbitrary redices, however, produces effects which cannot be captured by sequential transformations.[30]

[30] In [28], a typical example is presented which shows that these effects model properties of "truly parallel systems" in a natural way.

The investigations begin with a notion of parallel independence for two direct transformations. The Commutativity theorem proves that parallel independence implies that the result of the transformation is independent of the sequential order in which the two participating rules are applied.

**Definition 5.1** (*Parallel independence*). Two redices $m: L \to G$ and $n: M \to G$ for the transformation rules $r: L \to R$ and $s: M \to S$, respectively, are *parallel-independent* if they overlap in gluing items only, i.e. $m(L) \cap n(M) \subseteq m(r \triangledown m) \cap n(s \triangledown n)$.

**Theorem 5.2** (Commutativity of direct transformations). *If $m: L \to G$ and $n: M \to G$ are redices in the object $G$ for the rules $r: L \to R$ and $s: M \to S$, respectively, there are redices $p = s_n \circ m: L \to s_n(G)$ and $q = r_m \circ n: M \to r_m(G)$ such that $r_p(s_n(G)) = s_q(r_m(G))$ if and only if the redices $m$ and $n$ are parallel-independent.*

**Proof.** First suppose $m$ and $n$ are parallel-independent. Consider Diagram 12. Square (1) depicts the direct transformation of $G$ with $r$ at $m$ and square (2) the direct transformation of $G$ with $s$ at $n$. For $p = s_n \circ m$ and $q = r_m \circ n$ to be redices, it is to show that they are total morphisms which means to show (i) $m(L) \subseteq G_{s_n}$ and (ii) $n(M) \subseteq G_{r_m}$. We explicitly show (i); the argument for (ii) is symmetrical.

Suppose $o \notin G_{s_n}$. By Construction 2.6, it implies either $o \in \overline{n(s \triangledown n)}$[31] or there is a term $t \in T^{\mathrm{Sig}}(x)$ such that $(*)\, t^G(o) \in \overline{n(s \triangledown n)}$.[32] The first case implies $o \notin m(L)$ because $m$ and $n$ parallel-independent. The second case implies $o \notin m(L)$, too: Since $m$ is a redex, it is a total morphism and its image in $G$ is a subalgebra of $G$. Thus, the assumption $o \in m(L)$ implies $t^G(o) \in m(L)$ which is a contradiction to the parallel independence of $m$ and $n$ [compare $(*)$]}. Therefore, if $o \notin G_{s_n}$, $o \notin m(L)$, which immediately provides that $s_n \circ m$ is total.

Hence, the existence of the redices $p$ and $q$ is guaranteed and we must prove $r_p(s_n(G)) = s_q(r_m(G))$. For this purpose, let square (3) in Diagram 12 be constructed as

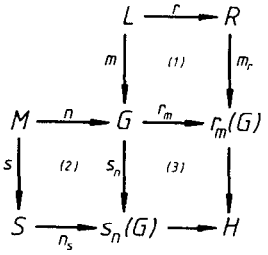


Diagram 12. Parallel independence and the commutativity property.

[31] $\overline{(s \triangledown n)}$ is a short notation for $L - (s \triangledown n)$.

[32] Note that Sig is the underlying graph structure according to General assumption 3.8.

the pushout of $r_m$ and $s_n$. Now (1)+(3) is the pushout diagram reflecting the direct transformation of $s_n(G)$ with $r$ at $p$ and (2)+(3) reflects the direct transformation of $r_m(G)$ with $s$ at $q$. The uniqueness of the pushout construction provides that the pushout objects of (3), (1)+(3), and (2)+(3) are isomorphic which completes this part of the proof.

Conversely, suppose $p=s_n\circ m$ and $q=r_m\circ n$ are redices, i.e. total morphisms. Then, $n(M)\subseteq G_{r_m}$ and $m(L)\subseteq G_{s_n}$. By Construction 2.6, we conclude $n(M)\subseteq\overline{m(L-r\triangledown m)}$ and $m(L)\subseteq\overline{n(M-s\triangledown n)}$. By construction of $r\triangledown m$ and $s\triangledown n$, this results in

$$(*)\qquad n(M)\cap m(L)\subseteq[\overline{m(L)}\cup m(r\triangledown m)]\cap[\overline{n(M)}\cup n(s\triangledown n)].$$

But, obviously, we have that:

(1) $[n(M)\cap m(L)]\cap[\overline{m(L)}\cap\overline{n(M)}]=\emptyset$.

(2) $[n(M)\cap m(L)]\cap[\overline{m(L)}\cap n(s\triangledown n)]=\emptyset$.

(3) $[n(M)\cap m(L)]\cap[m(r\triangledown m)\cap\overline{n(M)}]=\emptyset$.

Thus, (*) implies $n(M)\cap m(L)\subseteq m(r\triangledown m)\cap n(s\triangledown n)$. □

Hence, parallel independence of two rules implies local confluency. Moreover, the effect of applying two parallel-independent rules in any order can be obtained by a single direct transformation if the parallel composition of the two rules is used.

**Definition 5.3** (*Parallel rule and parallel redex*). If $r:L\to R$ and $s:M\to S$ are two transformation rules, the *parallel rule* $r+s$ is defined as the disjoint union of $r$ and $s$, i.e. $r+s=r\uplus s:L\uplus M\to R\uplus S$.

If $RS$ is a rule system, $RS^{\mathrm{P}}$ is the *parallel closure* of $RS$ which exactly contains $RS$ and all parallel rules which can be built within $RS^{\mathrm{P}}$.

The *parallel redex* $m+n$ for two redices $m:L\to G$ and $n:M\to G$ is defined by: $m+n:L\uplus M\to G$ such that $m+n(x)=m(x)$ if $x\in L$ and $m+n(x)=n(x)$ if $x\in M$.

**Theorem 5.4** (Parallel independence and parallel rule). *If redices $m:L\to G$ and $n:M\to G$ for the transformation rules $r:L\to R$ and $s:M\to S$, respectively, are parallel-independent, the application of the parallel rule $r+s$ at the parallel redex $m+n$ to $G$ results in the same object as any sequential application of $r$ and $s$, i.e. $s_{(r_m\circ n)}(r_m(G))=r_{(s_n\circ m)}(s_n(G))=(r+s)_{(m+n)}(G)$.*

**Proof.** Consider again Diagram 12. Note that the result of the sequential application of $r$ and $s$, i.e. the object $H$, has been constructed as the colimit of $s$, $n$, $m$, and $r$.[33] These morphisms make up the boldface part of Diagram 13. $L+M$ and $R+S$ are the colimits (coproducts) of $L$ and $M$ and of $R$ and $S$, respectively. The morphisms $i_1$–$i_4$

[33] The colimit of a diagram is unique up to isomorphism; cf. [21]. Since we do not distinguish objects if they are isomorphic, the colimit of a diagram is unique in our framework. Note that due to Corollary 2.10, the underlying category of graph structures and partial homomorphisms is finitely co-complete.

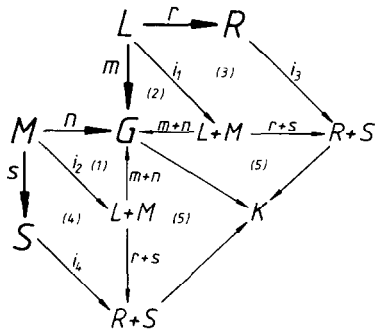


Diagram 13. Parallel rule and parallel-independent redices.

are the universal embeddings. The parallel rule $r+s$, as it is constructed in Definition 5.3, coincides with the universal morphism for coproducts such that the subdiagrams (3) and (4) commute.[34] Analogously, the parallel redex $m+n$ is the universal completion such that subdiagrams (1) and (2) commute. (5) is the pushout diagram reflecting the direct transformation of $G$ with $r+s$ at $m+n$.

Thus, Diagram 13 commutes and is thereby a cocone for the boldface part. Since it has been constructed as a composition of partial colimits, it is also a colimit of the boldface part. Uniqueness of colimits immediately provides that $K$ coincides with $H$ which is the colimit of the boldface diagram constructed in the proof of Theorem 5.2; compare Diagram 12. □

The converse of Theorem 5.4 is not true: Applicability of the parallel rule at an arbitrary redex $p$ does not imply that $p$ can be decomposed into parallel-independent redices for the components of the parallel rule.

**Example 5.5** (*Parallel rule and dependent redices*). Consider again the rule in Fig. 4. It is a parallel rule which is applied in Fig. 7 at a redex which is not d-injective. Obviously, the redices for the component rules (i.e. deletion of $q$ and addition of father relation) are not independent; cf. Definition 5.1.

Example 5.5 demonstrates that the addition of parallel rules to a given rule system can increase the possible transformations and the set of objects which can be generated from some start object. The results of the classical approach can be generalized to the single-pushout framework if redices are restricted to d-injective ones.

**Proposition 5.6** (Parallel rule and parallel independence). *If the parallel rule* $r+s: L+M \to R+S$ *is applicable to* $G$ *at a d-injective redex* $p: L+M \to G$, *its*

[34] The coproduct $A+B$ in $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$ can be constructed as the pushout of $\emptyset: \emptyset \to A$ and $\emptyset: \emptyset \to B$. The universal morphisms are then given by Construction 2.6.

*decomposition to the components, i.e.* $m=p_{|L}: L\to G$ *and* $n=p_{|M}: M\to G$, *is a pair of parallel-independent redices for the rule* $r: L\to R$ and $s: M\to S$, *respectively; therefore,* $s_{(r_m\circ n)}(r_m(G))=r_{(s_n\circ m)}(s_n(G))=(r+s)_p(G)$.

**Proof.** If $p$ is d-injective, $p(x)=p(y)$ implies $x=y$ or $x, y\in(r+s)\triangledown p$ and by Proposition 3.11, $(r+s)\triangledown p=(L+M)_{(r+s)}=L_r+M_s$. Thus, $x\in L$, $y\in M$, and $m(x)=n(y)$ implies $x\in L_r$ and $y\in M_s$ which, again by Proposition 3.11, means $x\in r\triangledown m$ and $y\in s\triangledown n$. Hence, parallel independence of $m$ and $n$ is guaranteed which, by Theorem 5.4, immediately proves the second part of the proposition. □

**Corollary 5.7** (Generated language). *If RS is a rule system,* $RS^{\mathrm{P}}$ *its closure w.r.t. parallel rules, and G an arbitrary start object,*

(1) $RS(G)\subseteq RS^{\mathrm{P}}(G)$,

(2) $RS(G)\neq RS^{\mathrm{P}}(G)$ *for some rule systems, and*

(3) *RS generates the same language as* $RS^{\mathrm{P}}$ *if redices are restricted to d-injective ones, i.e.* $RS_{\mathrm{i}}(G)=RS^{\mathrm{P}}_{\mathrm{i}}(G)$.

**Proof.** (1) is obvious since $RS\subseteq RS^{\mathrm{P}}$. (2) can easily be shown by e.g. Example 5.5. (3) is an immediate consequence of Proposition 5.6 and of the facts that d-injectivity of $m+n$ w.r.t. $r+s$ implies d-injectivity of $m$ and $s_n\circ m$ w.r.t. $r$ or of $n$ and $r_m\circ n$ w.r.t. $s$. The proof is straightforward. □

## 6. Amalgamation

Sequential and parallel composition of rules is a device to integrate the effects of several rules into a single one. Therefore, all results concerning this kind of composition are statements of equivalence expressing that there is a one-to-one correspondence between transformations with or without composed rules. The situation is different if we consider gluing of rules, called amalgamation. This concept has been introduced in [4] as a synchronization device for graph transformation systems which model the behavior of distributed systems. The work in [4] has been motivated by Degano and Montanari [7], who first introduced the idea of rule gluing and gave an explicit operational description.

This section reflects the theory presented in [4] for the single-pushout approach. We focus on theoretical aspects and refer to [14] for examples. All theorems of this section require redices to be d-injective.[35]

**General assumption 6.1** (*Redices*). All redices in this section are d-injective.

The key to amalgamation is the notion of *subrule* and *remainder*.

[35] D-injectivity is a sufficient condition for the theorems. It is not necessary in most cases. It is left to future research to investigate amalgamations at arbitrary redices.

**Definition 6.2** (*Subrule and remainder*). A rule $t: N \to T$ is a *subrule* of a rule $r: L \to R$ if there are two *total* morphisms $i: N \to L$ and $j: T \to R$ such that (1) $j \circ t = r \circ i$ and (2) $i$ is a d-injective redex for $t$. The $(i, j)$-*remainder* of $r$ w.r.t. $t$ is the rule $r -_{(i,j)} t: P \to R$ which is defined in Diagram 14 as the unique morphism for the pushout (1) of $t$ and $i$ such that $(r -_{(i,j)} t) \circ i_t = j$ and $(r -_{(i,j)} t) \circ t_i = r$.

We write $r - t$ for the remainder if the embeddings are obvious from the context. The subrule structure of a rule enables the decomposition of direct transformations.

**Theorem 6.3** (Subrule). *If $t: N \to T$ is a subrule of $r: L \to R$ with the embeddings $i: N \to L$ and $j: T \to R$ and $m: L \to G$ is a redex for $r$, there are redices $p$ and $q$ for $t$ and $r - t$, respectively, such that the direct transformation of $G$ with $r$ at $m$ can be decomposed into two direct transformations with $t$ at $p$ and with $r - t$ at $q$.*

**Proof.** Consider Diagram 15. The square (1) is the pushout constructed for the remainder in Definition 6.2; (2) + (3) is the pushout reflecting the direct transformation of $G$ to $H$ with $r$ at $m$. Since $r = (r - t) \circ t_i$, (2) + (3) can be decomposed into two pushout diagrams (2) and (3). Compositionality of pushouts guarantees that the diagram (1) + (2) reflects a direct transformation of $G$ to $K$ with $t$ at $p = m \circ i$. The morphism $p$ is total and d-injective since $m$ and $i$ are (cf. General assumption 6.1). If $q: P \to K$ is total and d-injective, (3) is the required direct transformation from $K$ to $H$ with $r - t$. D-injectivity of $m$ means: $m(x) = m(y)$ implies $x = y$ or $x, y \in L_r$. Since $r = (r - t) \circ t_i$, $L_r \subseteq L_{t_i}$ and $m(x) = m(y)$ implies $x = y$ or $x, y \in L_{t_i}$. Hence, $m$ is d-injective w.r.t. $t_i$ and Proposition 3.11(1) guarantees that $q$ is total. Moreover, $q$ is d-injective w.r.t. $r - t$. Square (2) is pushout and Corollary 2.8(2) states that $\ker(q) \subseteq t_i(\ker(m))$. But $t_i(\ker(m)) \subseteq t_i(L_r)$ since $m$ is d-injective. The remainder $r - t$ is constructed as a universal morphism such that $P_{(r-t)} = i_t(T_j) \cup t_i(L_r)$; cf. proof of Theorem 2.7. Thus, $\ker(q) \subseteq P_{(r-t)}$ stating d-injectivity of $q$ w.r.t. $r - t$. □

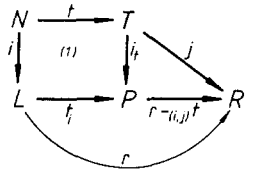


Diagram 14. Construction of Remainder

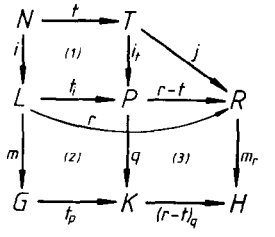


Diagram 15. Transformation decomposition.

The rest of the section considers the *synchronized behavior* of two rules which share a common subrule $t$. In a synchronized behavior, the effect of the shared rule $t$ is produced only once, which models a *handshake* at $t$.

**Definition 6.4** (*Related rules, amalgamable redices, synchronized effect*). Two rules $r: L \to R$ and $s: M \to S$ are related w.r.t. a third rule $t: N \to T$ if $t$ is a subrule of $r$ and $s$. In this case, we say that $r$ and $s$ are *t-related.*

Let $(i,j): t \to r$ and $(e,f): t \to s$ be the corresponding embeddings. Two redices $m: L \to G$ and $n: M \to G$ for $r$ and $s$, respectively, are *t-amalgamable* if $m \circ i = n \circ e$ and $m(L) \cap n(M) \subseteq m \circ i(N) \cup [m(r \triangledown m) \cap n(s \triangledown n)]$.

Application of the subrule $t$ at $m \circ i = n \circ e$ produces an object $X$ and induced redices $p$ and $q$ in $X$ for the remainders $r - t$ and $s - t$, respectively; cf. Theorem 6.3. The *t-synchronized effect* $r_m \|_t s_n: G \to H$ is defined to be the transformation from $G$ via $X$ to $H$ given by $t_{m \circ i}: G \to X$ and $((r-t)+(s-t))_{p+q}: X \to H$.

**Proposition 6.5** (Synchronized effect). *Let $r$ and $s$ be t-related and $m$ and $n$ be t-amalgamable redices for them, the induced redices $p$ and $q$ for the remainders as defined in Definition* 6.4 *are parallel-independent.*

**Proof.** The whole situation is depicted in Diagram 16. Diagram (2)+(3) is the direct transformation of $G$ with the rule $t$ at the redex $n \circ e$. Diagram (5)+(4) reflects the direct transformation of $G$ with $t$ at $m \circ i$. The redices $m$ and $n$ are amalgamable such that $m \circ i = n \circ e$. Therefore, the diagrams (2)+(3) and (5)+(4) depict the same pushout which is indicated in the diagram by the fact that the diagrams (2)+(3) and (5)+(4) overlap in the morphism $a$. (2) is the transformation of $M$ with $t$ at $e$ and (5) the transformation of $L$ with $t$ at $i$ according to Definition 6.2.

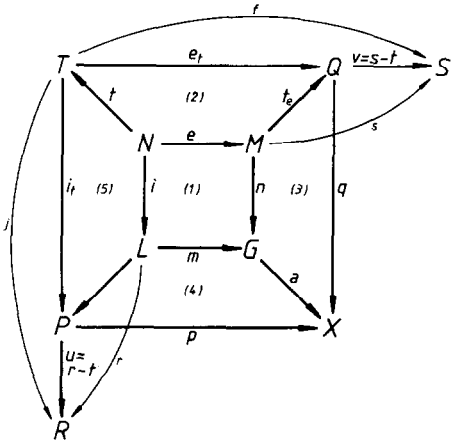


Diagram 16. Remainder redices in synchronized effect.

Parallel independence of $p$ and $q$, i.e. $p(P) \cap q(Q) \subseteq p(u \triangledown p) \cap q(v \triangledown q)$, remains to be shown. Since $p$ and $q$ are d-injective, we must prove that

$$p(x) = q(y) \text{ implies } x \in P_u \text{ and } y \in Q_v.$$

The morphism $u = r - t$ and $v = s - t$ are constructed as universal morphisms in Definition 6.2 and $j$ and $f$ are total such that

$$(*) \qquad P_u = i_t(T) \cup t_i(L_r) \quad \text{and} \quad Q_v = e_t(T) \cup t_e(M_s).$$

Since $P$ and $Q$ are pushout objects, $i_t$ and $t_i$ and $e_t$ and $t_e$ are jointly surjective, respectively, such that there are four cases to be considered:

*Case 1*: $x \in i_t(T)$ and $y \in e_t(T)$: (*) immediately implies that $x$ and $y$ are gluing points.

*Case 2*: $x \in i_t(T)$ and $y \in t_e(M)$: $x \in i_t(T)$ means that there exists $z \in T$ such that $i_t(z) = x$. Since diagram (2)+(3) coincides with diagram (5)+(4), $p \circ i_t = q \circ e_t$. Thus, $q \circ e_t(z) = q(y)$. If $e_t(z) = y$, $y$ has a preimage w.r.t. $e_t$ and is gluing item by (*). If $e_t(z) \neq y$, $q$ identifies $y$ and $e_t(z)$ which implies that $y$ is gluing item since $q$ is d-injective.

*Case 3*: $x \in t_i(L)$ and $y \in e_t(T)$: Follows from an argument similar to case 2.

*Case 4*: $x \in t_i(L)$ and $y \in t_e(M)$: In this case there exists $c \in L$ such that $t_i(c) = x$ and there exists $d \in M$ with $t_e(d) = y$. Since the subdiagrams (3) and (4) commute, $p(x) = a \circ m(c)$ and $q(y) = a \circ n(d)$.

Suppose as a first case $m(c) = n(d)$. Since the redices $m$ and $n$ are amalgamable and d-injective either $c$ and $d$ are gluing point w.r.t. $r$ and $s$, respectively, or there exists $b \in N$ with $i(b) = c$ and $e(b) = d$. The former immediately results in $x$ and $y$ being gluing points by (*). The latter results in $x = t_i \circ i(b)$ and $y = t_e \circ e(b)$ which implies that $x$ and $y$ have preimages w.r.t. $i_t$ and $e_t$, respectively, and we are done.

Suppose as the second case $m(c) \neq n(d)$. Then $m(c), n(d) \in \ker(a)$. This implies that both $c$ and $d$ have preimages w.r.t. both morphisms $n$ and $m$ because (3) and (4) are pushouts [cf. Corollary 2.8(2)]. These preimages must be elements of the kernel of $t_e$ and $t_i$. Thus, they must have preimages w.r.t. $n \circ e$ and $m \circ i$ and we are back to the arguments in cases 1–3, which completes the proof. □

Proposition 6.5 shows that the synchronized effect of two rules as defined in Definition 6.4 models shared behavior *exactly* on the part affected by the shared rule. The *local* operational effects of both rules, i.e. those parts not in the subrule, are independent.

The synchronized effect of two rules can be obtained by simple direct transformations if *amalgamated rules* are constructed and applied.

**Definition 6.6** (*Amalgamated rule*). If $r: L \to R$ and $s: M \to S$ are $(t: N \to T)$-related via embeddings $(i, j): t \to r$ and $(e, f): t \to s$, respectively, the *amalgamated rule* $r \oplus_t s: U \to V$ is constructed in Diagram 17. $U$ and $V$ are pushouts of $i$ and $e$ and of $j$ and $f$, respectively. $r \oplus_t s$ is the unique morphism such that $(r \oplus_t s) \circ e_i = f_j \circ r$ and $(r \oplus_t s) \circ i_e = j_f \circ s$.

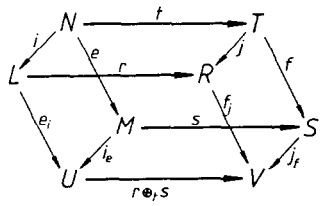


Diagram 17. Construction of the amalgamated rule.

Note that $r \oplus_t s$ is the colimit of $(i, j)$ and $(e, f)$ in the category of arrows over $\mathrm{Alg}^{\mathrm{P}}(\mathrm{Sig})$. The short notation $r \oplus_t s$ for the amalgamated rule is not precise because the result of the amalgamation construction depends on the actual embeddings. Thus, we assume in the sequel that the involved embeddings are obvious from the context.

**Theorem 6.7** (Amalgamation). *There are amalgamable redices $m$ and $n$ for the rules $r: L \to R$ and $s: M \to S$ such that the corresponding synchronized effect transforms $G$ to $H$, i.e. $r_m \|_t s_n: G \to H$ if and only if there is a redex $o$ for the amalgamated rule $r \oplus_t s$ such that $(r \oplus_t s)_0: G \to H$ is a direct transformation.*

**Proof.** Suppose $(r \oplus_t s)_0: G \to H$ is a direct transformation. Construct the redices $m: L \to G$ and $n: M \to G$ by defining $m = o \circ e_i$ and $n = o \circ i_e$.

They are both d-injective: $i$ and $e$ are d-injective w.r.t. $t$ such that $\ker(e_i) \subseteq i(\ker(e)) \subseteq i(N_t)$ and $\ker(i_e) \subseteq e(\ker(i)) \subseteq e(N_t)$. Since $t$ is a subrule of $r$ and $s$, $r \circ i = j \circ t$ and $s \circ e = f \circ t$. Thus, $e(N_t) \subseteq M_s$ and $i(N_t) \subseteq L_r$. Therefore, $i_e$ and $e_i$ are d-injective w.r.t. $s$ and $r$, respecively. The redex $o$ is d-injective and $U_{(r \oplus_t s)} = e_i(L_{(j_f \circ r)}) \cup i_e(M_{(j_f \circ r)}) = e_i(L_r) \cup i_e(M_s)$. Hence, $o \circ e_i$ and $o \circ i_e$ are d-injective w.r.t. $r$ and $s$.

They are also amalgamable since $m(x) = n(y)$ implies $o \circ e_i(x) = o \circ i_e(y)$. If $e_i(x) = i_e(y)$, $e_i(x)$ must be an image of $e_i \circ i$ since $U$ is a pushout of $e$ and $i$. But then $m(x) \in m \circ i(N)$. If $e_i(x) \neq i_e(y)$, $o$ identifies two different items which implies $x, y \in U_{(r \oplus_t s)} = e_i(L_r) \cup i_e(M_s)$. Hence, $x \in L_r$ and $y \in M_s$ due to d-injectivity of $e_i$ and $i_e$. Therefore, $m(x) \in m(L_r) \cup n(M_s) = m(r \triangledown m) \cup n(s \triangledown n)$.

For the implication in the reverse direction, amalgamable, d-injective redices $m$ and $n$ are given. We have to show that $o: U \to G$ constructed as the unique morphism such that $o \circ e_i = m$ and $o \circ i_e = n$ is d-injective. The proof is routine. It can be achieved straightforward by a complete case analysis for $x$ and $y$ if $o(x) = o(y)$ is given.

That the synchronized effect $r_m \|_t s_n: G \to H$ coincides with the direct transformation $(r \oplus_t s)_o: G \to H$ is a direct consequence of the fact that the synchronized effect of $r$ and $s$ is the colimit of the left part in Diagram 18 (cf. Definition 6.4 and Proposition 6.5) and direct transformations with amalgamated rules are defined to be colimits of the right part in Diagram 18. Since $o \circ e_i = m$ and $o \circ i_e = n$ and $U$ is a colimit itself, both colimits coincide up to isomorphism; cf. [21]. □

Note that Proposition 5.6 and Theorem 5.4 are special cases of Theorem 6.7 since parallel rules are amalgamations w.r.t. the empty shared subrule.

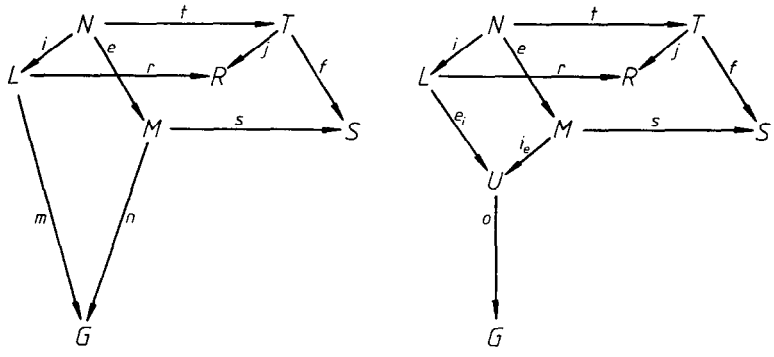


Diagram 18. Amalgamation is colimit construction.

## 7. Conclusion

The single-pushout approach to graph transformation presented in this article emerged from the observation that a transformation rule in the double-pushout framework [8] can be interpreted as a partial morphism in an appropriate category.

The rigorous investigations of algebraic categories with partial homomorphisms in Section 2 led to the notion of graph structures. It is exactly these *graph-like structures* which are closed w.r.t. finite colimits. These results seem to be analogous to the results in [13], where a detailed analysis of graph pushouts in the total case is provided. Ehrig and Kreowski [13] show that pushouts of graphs have certain properties which do not hold in arbitrary categories. These special properties are reflected in the partial case by the incompleteness w.r.t. colimits if the objects considered do not resemble graphs. Some hints that there is a tied connection between categories of total and partial homomorphisms are also given by [37, 24]. Future research shall focus on the details of this connection for algebraic categories.

Many results known from the double-pushout framework can be generalized if the transformation process is based on partial pushouts. A typical example is the embedding of transformation sequences. In the new approach, it is always possible to replay a transformation sequence in bigger contexts. On the other hand, the more general applicability of single-pushout rules produces new effects w.r.t. composition. For example, a transformation with parallel rules cannot always be decomposed into transformations with the components of the parallel rule as it is possible in the double-pushout setting; compare Section 5. Analogous results hold for amalgamated rules (Section 6). The one-to-one correspondence between composition of rules and composition of transformations can only be reobtained if the redices for the rules are properly restricted. The central properties in this respect are conflict-freeness and d-injectivity. With d-injective redices, *all* results of the algebraic approach carry over to the new framework. D-injectivity is the analog of the *identification* condition known from double-pushout transformations. These results disclose an asymmetry of the

gluing conditions *dangling* and *identification* in the double-pushout approach. While the identification condition is cruicial for almost all results about composition of rules and transformations, the dangling condition can easily be dropped in the operational semantics without changing the statements of the central results.

Not only many results can be generalized for single-pushout transformations, but the proofs are also shorter, the constructions simpler, and there are less technical side conditions. A typical example is the notion of a subrule which needs no technical extra requirements at all in the approach presented above. The technical easiness on the level of direct transformation and rule composition, however, has been purchased by more complex constructions on the fundamental categorical level of morphisms and pushouts. Some of the aspects which have to be handled explicitly on the level of rules and transformations in the double-pushout approach have been hidden in the basic constructions of the new framework. Future research must show if the new level of abstraction is sound w.r.t. further extensions of the theory, for example w.r.t. distributed transformations.

Many of the new effects which can be observed in the general single-pushout approach are due to redices which are not d-injective or conflict-free. These redices are able to model certain aspects of amalgamation. Amalgamation and redices which are not d-injective are both models for rule applications overlapping in nongluing items. The precise relationship between these two concepts needs further theoretical investigations. An important question in this respect is whether redices can be restricted to injective ones if arbitrary amalgamation of rules is admitted, i.e. can noninjective redices or all interesting noninjective redices be modeled by amalgamation? A positive answer to this question would be very valuable. It would reduce all concepts which model aspects of parallelism to a single, central one, i.e. amalgamation.

From the practical point of view, implementations of graph transformation systems should be available in order to prove the usefulness of graph-rewriting methods in system design. We are currently developing a prototype system based on single-pushout transformations at the Technical University of Berlin; cf. [2]. Experiments with this system shall show which extensions of the pure single-pushout approach presented in this article are necessary for practical applications and software engineering (for example, a concept for variables or attributes).

## Appendix A. Basic notions of the double-pushout approach

The algebraic graph grammar approach of [8] is based on the category of Labeled Graphs, compare Example 3.3, and *total* morphisms, i.e. Alg(Labeled Graphs).

**Definition A.1** (*Graph transformation rule*). A *graph transformation rule* $p=(l: K \to L, r: K \to R)$ consists of two graph morphisms $l$ and $r$ from the *gluing graph* $K$ to the *left-hand side* $L$ and to the *right-hand side* $R$, respectively. The left-hand morphism $l$ is required to be injective.

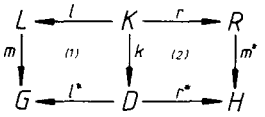


Diagram 19. Direct transformation (classical approach).

**Definition A.2** (*Direct transformation*). A graph $G$ can be *directly transformed* to another graph $H$ using rule $p=(l: K\to L, r: K\to R)$ if there are two diagrams (1) and (2) (cf. Diagram 19) which are pushout diagrams in the category of graphs and graph morphisms.[36]

In a direct transformation situation, the morphism $m: L\to G$ is called a *redex* or *match* of $p$ in $G$. The constructive version of the direct transformation builds the *context graph* $D$ and the result graph $H$ if a redex is given. For the description of the construction, we need some operations on graphs.

**Definition A.3** (*Operations on graphs*). For graphs $G$ and $H$, $G+H$ denotes the disjoint union of $G$ and $H$, i.e. the disjoint union of the vertex and edge sets with the operations $s^{G+H}$ given by $s^{G+H}(x)=s^G(x)$ if $x\in G$ and $s^{G+H}(x)=s^H(x)$ otherwise and $t^{G+H}$, defined by the same scheme.

If $H$ is a graph and $(V, E)$ a pair of subsets of its vertices and its edges, $H-(V, E)$ denotes the largest subgraph of $H$ whose vertex and edge sets are contained in $H_V-V$ and $H_E-E$, respectively, i.e.

$$(H-(V, E))_V=H_V-V,$$

$$(H-(V, E))_E=\{e\in H_E-E \mid s^H(e), t^H(e)\in(H-(V, E))_V\}.$$

$s^{H-(V,E)}$, $t^{H-(V,E)}$, and the labeling functions are the restrictions of the corresponding functions of $H$ to the smaller vertex and edge sets of $H-(V, E)$.

If $\sim=(\sim_V, \sim_E)$ is a pair of relations on the vertex and edge sets of some graph $H$, $H_{/\sim}$ denotes the quotient of $H$ w.r.t. the smallest congruence which contains $\sim$. The corresponding natural morphism is denoted by $\sim: H\to H_{/\sim}$.

With these prerequisites, we can construct the result graph $H$ of a direct transformation from a graph $G$ using rule $p=(l: K\to L, r: K\to R)$ at redex $m: L\to G$.

**Construction A.4** (*Direct transformation*). The direct transformation of a graph $G$ with the rule $p=(l: K\to L, r: K\to R)$ at a redex $m: L\to G$ results in a graph $H$ which can be constructed in three steps:

(1) *Remove*: $D:=G-(m_V(L_V-l_V(K_V)), m_E(L_E-l_E(K_E)))$. Let $l_D: D\to G$ denote the obvious inclusion morphism.

(2) *Add*: $E:=D+R$. Let $i_D: D\to E$ and $i_R: R\to E$ denote the obvious inclusions.

(3) *Embed*: $H:=E_{/\sim}$, where $x\sim y$ if $x=i_R\circ r(z)$ and $y=i_D\circ(l_D)^{-1}\circ m\circ l(z)$.

[36] *The algebraic approach to graph transformation* [8, 9, 16] is based on this notion for direct transformations. Due to its form, it is called the *double-pushout approach*.

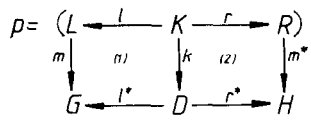


Diagram 20. Definition of direct transformation.

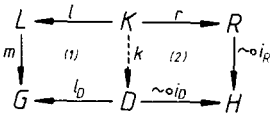


Diagram 21. Construction of a direct transformation.

**Theorem A.5** (Direct transformation). *The definition and the construction of direct transformations are related as follows:*

(1) *Let Diagram* 20 *represent a direct transformation of G to H with rule* $p=(l: K \to L, r: K \to R)$ *at redex* $m: L \to G$ *as defined in Definition A.2.*

(*a*) *H is uniquely determined by G, p, and m.*

(*b*) *m satisfies the gluing conditions dangling and identification* 2.

(*c*) *D and H coincide with the graphs constructed in Construction A.4.*

(2) *Let Diagram* 21 *depict a construction as in Construction A.4.*

(*a*) $k=(l_D)^{-1} \circ m \circ l: K \to D$ *is total if the gluing condition identification* 1 *is satisfied. In this case, subdiagram* (2) *is a pushout.*

(*b*) *If m satisfies dangling and identification* 2, *also diagram* (1) *is a pushout.*

*Identification* 1: $m(x)=m(y)$ *implies* $x, y \in l(K)$ *or* $x, y \notin l(K)$.

*Identification* 2: $m(x)=m(y)$ *implies* $x, y \in l(K)$ *or* $x=y$.

*Dangling*: $s_i(e)$ *or* $t_j(e) \in m_V(L_V - l_V(K_V))$ *for some* $e \in G_E$ *implies* $e \in m_E(L_E)$.

## Appendix B. Completion of partial morphisms

For a big class of graph structures, so-called *hierarchical graph structures*, partiality of a homomorphism can be modeled by a *total* homomorphism which maps undefined objects to some special $\perp$-items. The corresponding completion with the necessary $\perp$-structure is presented below. It relates the approach to single-pushout transformations presented above to the one in [5].

**Definition B.1** (*Hierarchical graph structures*). A graph structure $(\mathsf{S}, \mathsf{OP})$ is *hierarchical* if there is no infinite sequence $\mathsf{op}_1: \mathsf{s}_1 \to \mathsf{s}_2, \mathsf{op}_2: \mathsf{s}_2 \to \mathsf{s}_3, \ldots, \mathsf{op}_i: \mathsf{s}_i \to \mathsf{s}_{i+1}, \mathsf{op}_{i+1}: \mathsf{s}_{i+1} \to \mathsf{s}_{i+2}, \ldots$ of operator symbols.

If $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ is a hierarchical graph structure, the relation $\preceq$ on $\mathsf{S}$ defined by $s' \preceq s$ if $T^{\mathsf{Sig}}_{s \to s'} \neq \emptyset$ is a well-founded partial order. Thus, noetherian induction can be used if statements have to be proven for all sorts.

Note that all examples in Section 3.1 are hierarchical.

**Definition B.2** (*Junk completion*). If $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ is a hierarchical graph structure,

(1) $\mathrm{opns}(s)=(\mathrm{opns}(s)_i)_{i\in I_s}$ denotes the vector of operator symbols in OP which take arguments of sort $s$, i.e. for all $i\in I_s$, $\mathrm{opns}(s)_i=\mathsf{op}_i:\mathsf{s}\to\mathsf{s}_i$.

(2) $\mathrm{sorts}(s)=(\mathrm{sorts}(s)_i)_{i\in I_s}$ is the corresponding list of sort symbols such that $\mathrm{sorts}(s)_i$ is the value sort of $\mathrm{opns}(s)_i$ for all $i\in I_s$.

(3) the *junk completion* Junk(Sig) is the following equational specification, where $\vec{x}$ is a vector of variables such that $\vec{x}_i\in X_{\mathrm{sorts}(s)_i}$ for all $i\in I_s$:[37]

Junk(Sig)=
**Use** Sig
**Operations**
$(\bot_s:\mathrm{sorts}(s)\to s)_{s\in S}$
**Equations**
$(\mathrm{opns}(s)_i(\bot_s((\vec{x}))=\vec{x}_i)_{s\in S,\, i\in I_s}$

Note that in case of an infinite vector opns($s$) for some sort $s$, we have to handle signatures and algebras with operators which take infinitely many arguments. Hierarchical signatures, however, guarantee that we do not run into trouble with "infinitely deep" terms, i.e. each term w.r.t. the signature Junk(Sig) is of possibly infinite width but of finite height.

Junk(Sig)-algebras help in the analysis of $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. First, consider the relation of Alg(Sig) and $\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$. Obviously, they coincide on objects and each total homomorphism is a special partial one. Hence, $\mathrm{Alg}(\mathsf{Sig})\subseteq\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$, and with the aid of Junk(Sig), the inclusion $\subseteq:\mathrm{Alg}(\mathsf{Sig})\to\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ turns out to be a left adjoint functor.

**Proposition B.3** (Inclusion functor). *If* Sig *is a hierarchical graph structure, the inclusion* $\subseteq:\mathrm{Alg}(\mathsf{Sig})\to\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ *is a left adjoint functor.*

**Proof.** We construct a mapping $P\mapsto(\mathrm{F}(P), \mathrm{u}^P:\mathrm{F}(P)\to P)$ which assigns to each object $P\in\mathrm{Alg}^{\mathrm{P}}(\mathsf{Sig})$ another object $\mathrm{F}(P)\in\mathrm{Alg}(\mathsf{Sig})$ and a partial homomorphism $\mathrm{u}^P$, and show that the so-defined mapping is a co-free construction, i.e. for each partial homomorphism $f:B\to P$, there is a unique total homomorphism $f^*:B\to\mathrm{F}(P)$ such that $\mathrm{u}^P\circ f^*=f$. The situation is depicted in Diagram 22.

Define $\mathrm{F}(P)=[\mathrm{Free}(P)]_{\mathsf{Sig}}$, where $\mathrm{Free}:\mathrm{Alg}(\mathsf{Sig})\to\mathrm{Alg}(\mathsf{Junk}(\mathsf{Sig}))$ is the free construction between equationally defined classes of algebras (cf. [15]) and

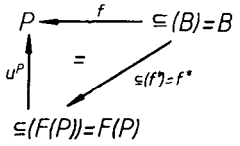


Diagram 22. Co-free situation.

[37] The phrase "**Use** Sig" in the specification below indicates that Sig is a subsignature of the defined signature.

$[\ ]_{\mathsf{Sig}}: \mathsf{Alg}(\mathsf{Junk}(\mathsf{Sig})) \to \mathsf{Alg}(\mathsf{Sig})$ is its right adjoint. Since Free is consistent, we obtain an injective universal homomorphism $\mathsf{v}^P: P \to [\mathrm{Free}(P)]_{\mathsf{Sig}}$ and define $\mathsf{u}^P = (\mathsf{v}^P)^{-1}$.

Now it needs to be proved that the assignment $P \mapsto (\mathrm{F}(P), \mathsf{u}^P: P \to \mathrm{F}(P))$ satisfies the co-universal property above. Let $f: B \to P$ be a partial homomorphism. Define $f^*: B \to \mathrm{F}(P)$ for all $s \in \mathsf{S}$ by $f^*_s(x) = \mathsf{v}^P_s \circ f_s(x)$ if $x \in B_{f_s}$ and $f^*_s(x) = \perp_s^{\mathrm{Free}(P)}(\vec{f}^*(\mathrm{opns}(s)^B(x)))$ otherwise.[38] $f^*$ is well-defined by definition. It is to be shown that it is totally defined and homomorphic. We use noetherian induction w.r.t. the partial order $\preceq$. If $s$ is a minimal element w.r.t. $\preceq$, there is no operator $\mathsf{op}: \mathsf{s} \to \mathsf{s}' \in \mathsf{OP}$. Hence, $\perp_s: \to \mathsf{s}$ is a constant which immediately implies that $f^*$ is total and homomorphic for $s$.

Now consider a nonminimal sort $s$. Since for all $s' \in \mathrm{sorts}(s): s' \preceq s$, $f^*$ is totally defined on all these sorts by induction hypothesis, we immediately obtain that $f^*$ is totally defined on $s$. It is homomorphic on $B_{f_s}$ since $f_s$ and $\mathsf{v}^P_s$ are and for $x \notin B_{f_s}$, we have that each operation $\mathsf{op}: \mathsf{s} \to \mathsf{s}' = \mathrm{opns}(s)_i: \mathsf{s} \to \mathsf{s}'$ for some $i \in I_s$. Since the operations satisfy the equations of $\mathsf{Junk}(\mathsf{Sig})$:

$$\begin{aligned}\mathsf{op}^{\mathrm{F}(P)}(f^*(x)) &= \mathsf{op}^{\mathrm{F}(P)}(\perp_s^{\mathrm{Free}(P)}(\vec{f}^*(\mathrm{opns}(s)^B(x)))) \\ &= \mathrm{opns}(s)_i^{\mathrm{F}(P)}(\perp_s^{\mathrm{Free}(P)}(\vec{f}^*(\mathrm{opns}(s)^B(x)))) \\ &= f^*(\mathrm{opns}(s)_i^B(x)) = f^*(\mathsf{op}^B(x)).\end{aligned}$$

The definition of $f^*$ immediately guarantees $B_{\mathsf{u}^P f^*} = B_f$ and $\mathsf{u}^P \circ f^* = (\mathsf{v}^P)^{-1} \circ f^* = (\mathsf{v}^P)^{-1} \circ \mathsf{v}^P \circ f = f$. $f^*$ is unique since each total $g: B \to \mathrm{F}(P)$ must coincide on $B_f$ with $f^*$ due to $\mathsf{u}^P \circ g = f$. Outside $B_f$, g must map $x$ homomorphically to elements outside $\mathsf{v}^P(P)$. There is exactly one such element, namely $\perp(\vec{f}^*(\mathrm{opns}(s)^B(x)))$.

The assignment on objects $P \mapsto \mathrm{F}(P)$ can be canonically extended to a functor $\mathrm{F}: \mathsf{Alg}^{\mathsf{P}}(\mathsf{Sig}) \to \mathsf{Alg}(\mathsf{Sig})$ which is right adjoint to $\subseteq: \mathsf{Alg}(\mathsf{Sig}) \to \mathsf{Alg}^{\mathsf{P}}(\mathsf{Sig})$. □

For hierarchical graph structures, the junk completion of Definition B.2 provides some further information about colimits and how they can be constructed if the following uniqueness constraint is additionally required in the completion process.

**Definition B.4** (*Sink completion*). If Sig is a hierarchical graph structure, its *sink completion* Sink(Sig) is the following specification with conditional equations:

Sink(Sig) =
**Use** Junk(Sig)
**Equations**
$(\mathrm{opns}(s)_i(y) = \perp_{s_i}(\vec{x}) \Rightarrow y = \perp_s(\mathrm{opns}(s)(y)))_{s \in S,\, i \in I_s}$

The categories $\mathsf{Alg}^{\mathsf{P}}(\mathsf{Sig})$ and $\mathsf{Alg}(\mathsf{Sink}(\mathsf{Sig}))$ are closely related by the following pair of functors.

[38] $\vec{f}^*$ is the extension of $f^*$ to vectors and $\mathrm{opns}(s)^B(x)$ is the vector such that for all $i \in I_s$, $\mathrm{opns}(s)^B(x)_i = \mathrm{opns}(s)_i^B(x)$.

**Definition and Proposition B.5** (*Completion functor*). If $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ is a hierarchical graph structure and $\mathsf{Sink}(\mathsf{Sig})$ its sink completion, the *completion functor* $\mathrm{T}: \mathsf{Alg}^{\mathrm{P}}(\mathsf{Sig}) \to \mathsf{Alg}(\mathsf{Sink}(\mathsf{Sig}))$ can be defined as follows:

(1) On objects, $\mathrm{T}(A)=\mathrm{Free}(A)$, where $\mathrm{Free}(A)$ is the free $\mathsf{Sink}(\mathsf{Sig})$-algebra over $A$.[39] Since $\mathrm{Free}: \mathsf{Alg}(\mathsf{Sig}) \to \mathsf{Alg}(\mathsf{Sink}(\mathsf{Sig}))$ is consistent, T can be chosen such that for all $s \in \mathsf{S}$, $A_s \subseteq \mathrm{T}(A)_s$.

(2) On morphisms, $\mathrm{T}(f: A \to B)=f^{\mathrm{T}}: \mathrm{T}(A) \to \mathrm{T}(B)$ is given for all $s \in \mathsf{S}$ by

$$f_s^{\mathrm{T}}(x)=\begin{cases} f_s(x) & \text{if } x \in A_f, \\ \bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(x))) & \text{otherwise.} \end{cases}$$

**Proof.** Noetherian induction on the sort relation $\preceq$ shows that $f^{\mathrm{T}}$ is a family of *total* mappings; compare proof of Proposition B.3. For $f^{\mathrm{T}}$ to be homomorphic, it is to be shown that

(1) $f^{\mathrm{T}}(\mathsf{op}^{\mathrm{T}(A)}(x))=\mathsf{op}^{\mathrm{T}(B)}(f^{\mathrm{T}}(x))$ for all $\mathsf{op} \in \mathsf{OP}$ and

(2) $f^{\mathrm{T}}(\bot^{\mathrm{T}(A)}(\vec{x}))=\bot^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\vec{x}))$ for all $\bot$-operators.

In the first case, two subcases can be distinguished, i.e. either $x \in A_f$ or $x \notin A_f$. If the former is true, $f^{\mathrm{T}}$ is homomorphic since $f$ is. If the latter is true, the definition of $f^{\mathrm{T}}$ provides: $\mathsf{op}^{\mathrm{T}(B)}(f_s^{\mathrm{T}}(x))=\mathsf{op}^{\mathrm{T}(B)}(\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(x))))$. Since $\mathsf{op}=\mathsf{opns}(s)_j$ for some $j \in I_s$, the equations in $\mathsf{Junk}(\mathsf{Sig})$ make sure that

$$\begin{aligned}\mathsf{op}^{\mathrm{T}(B)}(\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(x))))&=\mathsf{opns}(s)_j^{\mathrm{T}(B)}(\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(x)))) \\ &=f^{\mathrm{T}}(\mathsf{opns}(s)_j^{\mathrm{T}(A)}(x))=f^{\mathrm{T}}(\mathsf{op}^{\mathrm{T}(A)}(x)).\end{aligned}$$

In the second case, we have by construction of $\mathrm{T}(A)$ that $\bot^{\mathrm{T}(A)}(\vec{x}) \notin A$. Therefore, we obtain by definition of $f^{\mathrm{T}}$: $f_s^{\mathrm{T}}(\bot^{\mathrm{T}(A)}(\vec{x}))=\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(\bot^{\mathrm{T}(A)}(\vec{x}))))$. The equations of $\mathsf{Junk}(\mathsf{Sig})$ guarantee, for all $j \in I_s$, that: $\mathsf{opns}(s)_j^{\mathrm{T}(A)}(\bot^{\mathrm{T}(A)}(\vec{x}))=\vec{x}_j$. Hence, it can be concluded: $\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\mathsf{opns}(s)^{\mathrm{T}(A)}(\bot^{\mathrm{T}(A)}(\vec{x}))))=\bot_s^{\mathrm{T}(B)}(\vec{f}^{\mathrm{T}}(\vec{x}))$, which completes the proof that $f^{\mathrm{T}}: \mathrm{T}(A) \to \mathrm{T}(B)$ is a total homomorphism. By definition, the functor T preserves identities and respects morphism composition. □

**Definition and Proposition B.6** (*Restriction functor*). If $\mathsf{Sig}=(\mathsf{S}, \mathsf{OP})$ is a hierarchical graph structure and $\mathsf{Sink}(\mathsf{Sig})$ its sink completion, the *restriction functor* $\mathrm{P}: \mathsf{Alg}(\mathsf{Sink}(\mathsf{Sig})) \to \mathsf{Alg}^{\mathrm{P}}(\mathsf{Sig})$ can be defined as follows:

(1) On objects, $\mathrm{P}(A)_s=\{y \in A_s \mid y \neq \bot_s^A(\vec{x})\}$ for $s \in \mathsf{S}$ and for $\mathsf{op} \in \mathsf{OP}$, $\mathsf{op}^{\mathrm{P}(A)}=\mathsf{op}^A_{|\mathrm{P}(A)}$.

(2) On morphisms, $\mathrm{P}(f: A \to B)=f^{\mathrm{P}}: \mathrm{P}(A) \to P(B)$ is given by

$$f^{\mathrm{P}}(x)=\begin{cases} f(x) & \text{if } f(x) \in \mathrm{P}(B), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

**Proof.** The conditional equations of $\mathsf{Sink}(\mathsf{Sig})$ guarantee that $\mathrm{P}(A)$ is a Sig-algebra for each $A \in \mathsf{Alg}(\mathsf{Sink}(\mathsf{Sig}))$ as the following argument demonstrates: The assumption

[39] Free constructions for specifications with conditional equations always exist; cf. [32, 6]. The constructions of [6] can be easily extended to handle operators with infinitely many arguments.

$x \in P(A)_s$ and for some operation $op: s \to s'$, $op^{P(A)}(x) \notin P(A)_{s'}$ implies $op^A(x) = \perp^A_{s'}(\vec{y})$. But $A$ satisfies the equations in Sink(Sig). Hence, $x = \perp^A_s(\vec{z})$ and, therefore, $x \notin P(A)_s$ which is a contradiction to the assumption. By definition, $f^P$ is a family of partial mappings. The same argument as above provides that $x \in P(A)_{f^P}$ implies $op^{P(A)}(x) \in P(A)_{f^P}$ for all $op: s \to s'$. Thus, $P(A)_{f^P}$ is a subalgebra of $P(A)$. $f^P$ is homomorphic on its scope since $f$ is.

The definition of P immediately provides that identities and compositions of morphisms are preserved. □

**Theorem B.7** (Completion). *The categories* $Alg^P(Sig)$ *and* Alg(Sink(Sig)) *are equivalent for hierarchical graph structures* Sig.

**Proof.** The definition of the completion functor T and of the restriction functor P immediately provides $P \circ T = id$.

For the equivalence, it remains to be shown that $T \circ P \cong id$. The construction of $P(A)$ for a Sink(Sig)-algebra $A$ provides a total inclusion homomorphism $i: P(A) \to [A]$, where $[\ ]: Alg(Sink(Sig)) \to Alg(Sig)$ is the forgetful functor induced by the specification inclusion of Sig in Sink(Sig). The completion functor T is the free Sink(Sig)-construction on objects. Hence, we obtain a unique extension of $i$, i.e. a total homomorphism $i^*: Free(P(A)) \to A$ such that the Diagram 23 commutes. Note that $u^{P(A)}$ has been chosen to be the inclusion; cf. Definition B.5. By induction on the sort order $\preceq$, it can be shown that $i^*$ is an isomorphism. For a minimal sort $s$ w.r.t. $\preceq$, $\perp_s$ is a constant. Hence, $A_s = P(A)_s \uplus \{\perp^A_s\}$ and $i^*_s$ is surjective and injective.

Now consider a sort $s$ and assume that $i^*_{s'}$ is bijective for $s' \preceq s$. By Definition B.6,

$$A_s = P(A)_s \uplus \{\perp^A_s(\vec{x}) \mid \text{for all vectors } \vec{x} \text{ with } x_i \in A_{s_i} \text{ for } i \in I_s\}.$$

Thus, $A_s$ is generated by $P(A)_s$ and $i^*_s$ is surjective. For the proof that $i^*_s$ is injective, suppose that $i^*_s(x) = i^*_s(y)$ for some $x, y \in Free(P(A))_s$. If $i^*_s(x) \in P(A)_s$, we obtain $x, y \in P(A)_s$ and $x = y$ since Diagram 23 commutes. If $i^*_s(x) \notin P(A)_s$, we obtain $x = \perp^{Free(P(A))}(\vec{v})$, $y = \perp^{Free(P(A))}(\vec{w})$, and

$$\perp^A_s(\vec{i^*}(\vec{v})) = i^*_s(\perp^{Free(P(A))}(\vec{v})) = i^*_s(x) = i^*_s(y) = i^*_s(\perp^{Free(P(A))}(\vec{w})) = \perp^A_s(\vec{i^*}(\vec{w})).$$

Since $A$ satisfies the equations of Sink(Sig), it follows that, for all $k \in I_s$,

$$i^*(v_k) = opns(s)^A_k(\perp^A_s(\vec{i^*}(\vec{v}))) = opns(s)^A_k(\perp^A_s(\vec{i^*}(\vec{w}))) = i^*(w_k).$$

By induction hypothesis, $i^*_{s'}$ is injective for all sorts $s' \preceq s$. Hence, for all $k \in I_s$, $\vec{v}_k = \vec{w}_k$ and, therefore, $x = \perp^{Free(P(A))}(\vec{v}) = \perp^{Free(P(A))}(\vec{w}) = y$.

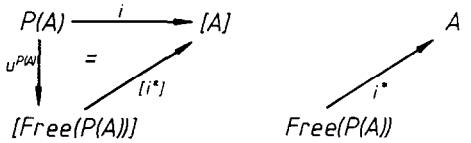


Diagram 23. Extension of restriction inclusion.

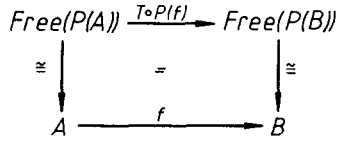


Diagram 24. Natural transformation for completion.

With this result on objects, it is obvious that Diagram 24 commutes for each total homomorphism $f$. □

Many graph structures which provide an algebraic model for some relational or graphical structures are hierarchical (compare examples in Section 3). For these graph structures, the whole theory presented in this article could have been described as a theory of sink-completed algebras due to Theorem B.7 (cf. [5]).

## References

[1] M. Arbib and E.G. Manes, *Arrows, Structures, and Functors* (Academic Press, New York, 1975).

[2] R. Arlt and M. Roeder, Grundlegende Datenstrukturen und Algorithmen zur Implementierung von algebraischen Graph Grammatiken, Studienarbeit, Fachbereich 20, Technische Universität Berlin, 1989.

[3] P. Boehm, H. Ehrig, U. Hummert and M. Löwe, Towards distributed graph grammars, in: H. Ehrig, M. Nagl, G. Rozenberg and A. Rosenfeld, eds., *Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 86–98.

[4] P. Boehm, H.-R. Fonio and A. Habel, Amalgamation of graph transformations: a synchronization mechanism, *J. Comput. System Sci.* **34** (1987) 377–408.

[5] P.M. van den Broek, Algebraic graph rewriting using a single pushout, in: *Proc. Internat. Joint Conf. on Theory and Practice of Software Development (TAP-SOFT '91)*, Lecture Notes in Computer Science, Vol. 493 (Springer, Berlin, 1991) 90–102.

[6] I. Claßen, Algebraische Grundlagen der Termersetzung mit bedingten Gleichungen, Tech. Report 88-04, Technische Universität Berlin, 1988.

[7] P. Degano and U. Montanari, A model of distributed systems based on graph rewriting, *J. ACM* **34** (2) (1987) 411–449.

[8] H. Ehrig, Introduction to the algebraic theory of graph grammars (a survey), in: *Proc. 1st Graph Grammar Workshop*, Lecture Notes in Computer Science, Vol. 73 (Springer, Berlin, 1979).

[9] H. Ehrig, Tutorial introduction to the algebraic approach of graph grammars, in: *Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 3–14.

[10] H. Ehrig, P. Boehm, U. Hummert and M. Löwe, Distributed parallelism of graph transformation, in: *Proc. 13th Internat. Workship on Graph-theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 314 (Springer, Berlin, 1988) 1–19.

[11] H. Ehrig, A. Habel, H.-J. Kreowski and F. Parisi-Presicce, From graph grammars to high-level replacement systems, in: *Proc. 4th Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1990) 269–291.

[12] H. Ehrig, M. Korff and M. Löwe, Tutorial introduction to the algebraic approach of graph grammars based on double and single pushouts, in: *Proc. 4th Internat. Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 24–37.

[13] H. Ehrig and H.-J. Kreowski, Pushout-properties: an analysis of gluing constructions for graphs, *Math. Nachr.* **91** (1979) 135–149.

[14] H. Ehrig and M. Löwe, Parallel and distributed derivations in the single pushout approach, Tech. Report 91/01, Department of Computer Science, Technical University of Berlin, 1991.
[15] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specifications* 1, Monographs in Computer Science (Springer, Berlin, 1985).
[16] H. Ehrig, M. Pfender and H.J. Schneider, Graph grammars: an algebraic approach, in: *Proc. 14th Ann. IEEE Symp. on Switching and Automata Theory* (1973) 167–180.
[17] J. Glauert, R. Kennaway and R. Sleep, A categorical construction for generalised graph rewriting, Tech. Report, School of Information Systems, University of East Anglia, Norwich NR4 7TJ, UK, 1989.
[18] A. Habel, Hyperedge replacement: grammars and languages, Ph.D. Thesis, University of Bremen, 1989.
[19] A. Habel and H.-J. Kreowski, May we introduce to you: hyperedge replacement, in: *Proc. 3rd Internat. Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 15–26.
[20] A. Habel, H.-J. Kreowski and D. Plump, Jungle evaluation, in: *Proc. 5th Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science, Vol. 332 (Springer, Berlin, 1988) 92–112.
[21] H. Herrlich and G. Strecker, *Category Theory* (Allyn and Bacon, Rockleigh, NJ, 1973).
[22] B. Hoffmann and D. Plump, Jungle evaluation for efficient term rewriting, in: J. Grabowski, P. Lescanne and W. Wechler, eds., *Proc. 1st Internat. Workshop on Algebraic and Logic Programming* (Akademie-Verlag, Berlin, 1988) 191–203.
[23] R. Kennaway, On "on graph rewriting", *Theoret. Comput. Sci.* **52** (1987) 37–58.
[24] R. Kennaway, Graph rewriting in some categories of partial maps, Tech. Report, University of East Anglia, 1990; also in: *Proc. 4th Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 532 (Springer, Berlin, 1991) 490–504.
[25] H.-J. Kreowksi, Is parallelism already concurrency? part 1: derivations in graph grammars, in: *Proc. 3rd Internat. Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 343–360.
[26] H.-J. Kreowski and G. Rozenberg, On structured graph grammars: Parts I and II, Tech. Report 3/88, Universität Bremen, 1988.
[27] H.-J. Kreowski and A. Wilharm, Is parallelism already concurrency? part 2: non-sequential processes in graph grammars, in: *Proc. 3rd Internat. Workshop on Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science, Vol. 291 (Springer, Berlin, 1987) 361–377.
[28] M. Löwe, Algebraic approach to graph transformation based on single pushout derivations with partial morphisms, Tech. Report 90/5, Department of Computer Science, Technical University of Berlin, 1990.
[29] M. Löwe, Implementing algebraic specifications by graph transformations, *J. Inform. Process. Cybernet. (EIK)* **26** (11/12) (1990) 615–641; also Tech. Report TU Berlin 89/26, Technical University of Berlin, Berlin, 1989.
[30] M. Löwe and H. Ehrig, Algebraic approach to graph transformation based on single pushout transformations, in: R.H. Möhring, ed., in: *Proc. 16th Internat. Workshop on Graph-theoretic Concepts in Computer Science*, Lecture Note in Computer Science, Vol. 484 (Springer, Berlin, 1991) 338–353.
[31] M. Löwe and R. Wilhelm, Risiken polizeilicher Datenverarbeitung, in: R. Kitzing, U. Linder and F. Obermaier, eds., *Schöne neue Computerwelt* (Verlag für Ausbildung und Studium (VAS) in der Elefantenpress, Berlin, 1988) 216–252.
[32] B. Mahr and J.A. Makowski, Characterizing specification languages which admit initial semantics, Tech. Report 232, Technion Haifa, 1982; also in *Theoret. Comput. Sci.* **31** (1984) 49–59.
[33] P. Padawitz, Graph grammars and operational semantics, *Theoret. Comput. Sci.* **19** (1982) 37–58.
[34] F. Parisi-Presicce, Modular system design applying graph grammar techniques, in: *ICALP'89*, Lecture Notes in Computer Science, Vol. 372 (Springer, Berlin, 1989) 621–636.
[35] D. Plump, Im Dschungel: Ein neuer Graph-Grammatik-Ansatz zur effizienten Auswertung rekursiv definierter Funktionen, Diplomarbeit, University of Bremen, 1986.
[36] J.C. Raoult, On graph rewriting, *Theoret. Comput. Sci.* **32** (1984) 1–24.
[37] E. Robinson and G. Rosolino, Categories of partial maps, *Inform. and Comput.* **79** (1988) 95–130.
[38] H.J. Schneider, Describing distributed systems by categorial graph grammars, in: *Proc. 15th Internat. Workshop on Graph-theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 411 (Springer, Berlin, 1990) 121–135.