Theoretical Computer Science 884 (2021) 23–43

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Single pushout rewriting in comprehensive systems of graph-like structures ☆

Patrick Stünkel [a,*], Harald König [b,a]

[a] Høgskulen på Vestlandet, Bergen, Norway
[b] FHDW Hannover, Germany

ARTICLE INFO

Article history:
Received 3 January 2021
Received in revised form 14 June 2021
Accepted 7 July 2021
Available online 15 July 2021

Keywords:
Single pushout rewriting
Partial morphism
Category theory
Hereditary pushout
Upper adjoint
Comprehensive system

ABSTRACT

The elegance of the *single-pushout* (SPO) approach to graph transformations arises from substituting total morphisms by partial ones in the underlying category. SPO's applicability depends on the durability of pushouts after this transition. There is a wide range of work on the question when pushouts exist in categories with partial morphisms starting with the pioneering work of Löwe and Kennaway and ending with an essential characterisation in terms of an exactness property (for the interplay between pullbacks and pushouts) and an adjointness condition (w.r.t. inverse image functions) by Hayman and Heindel.
Triple graphs and graph diagrams are frameworks to synchronise two or more updatable data sources by means of internal mappings, which identify common sub-structures. *Comprehensive systems* generalise these frameworks, treating the network of data sources and their structural inter-relations as a homogeneous comprehensive artefact, in which partial maps identify commonalities. Although this inherent partiality produces amplified complexity, we can show that Heindel's characterisation still yields existence of pushouts in the category of comprehensive systems and reflective partial morphisms and thus enables computing by typed SPO graph transformation.

1. Introduction and motivation

We dedicate this paper to Michael Löwe, the founder of the single-pushout approach [1] and simultaneously a pioneer in the investigation of categories of partial algebras with partial morphisms between them, cf. e.g. [2].

In this paper, we combine these two theories. We introduce the category of *comprehensive systems*, formally a category in which the inner structure of the objects can be described with partial maps, and will show that SPO rewriting is applicable in this category.

Comprehensive Systems have been introduced in [3] as a means for global consistency management. A comprehensive system represents a collection of inter-related software artefacts. Furthermore, they generalise other related formalisms such as *triple graphs* [4] and *graph diagrams* [5,6].

To provide an intuition of a comprehensive system (Definition 5 in Sect. 3), take a look at Fig. 1. It depicts an abstract representation (i.e. model) of the databases of three information systems run by a fictitious insurance company: There is a Contract Management System (CoM) D_1 that stores insurance contracts in an object database, a Case Management System

☆ In memory of *Michael Löwe*, 1956–2019.

* Corresponding author.
E-mail address: past@hvl.no (P. Stünkel).

https://doi.org/10.1016/j.tcs.2021.07.002

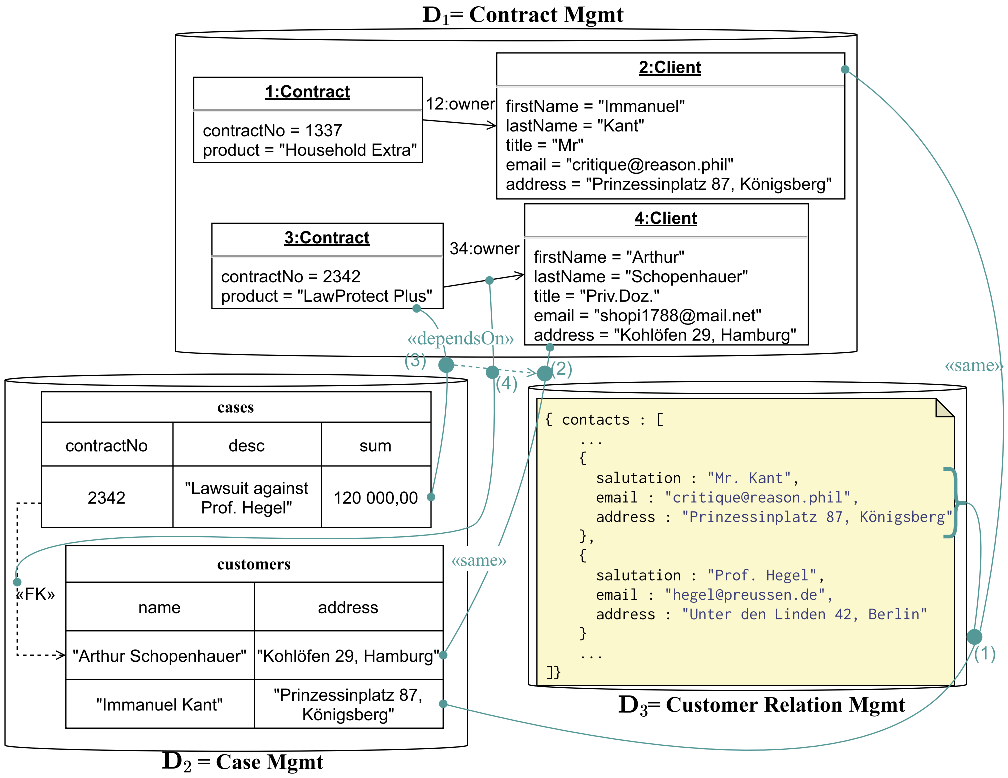


Fig. 1. Comprehensive system **D**: insurance information system databases. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

(CaM) D_2 storing information over active and resolved insurance claims in a relational database, and a Customer Relationship Management System (CRM) D_3 storing customer contacts in a JSON document database.

It is necessary to maintain global consistency of the databases' contents, especially in the presence of global *consistency rules* [7–9]. Let us assume the following consistency requirements in this scenario:

CR1 The `name`, `address`, and `email` information for every `customer/client/ contact` record, representing the *same* real-world person, stored in the three systems, must be unambiguous.
CR2 The existence of a `case` in D_2 *necessitates* the existence of a respective `contract` in D_1.

Violations of these constraint can only be discovered, if common and related elements in the systems are identified. Thus, one has to specify that the `client` record for *"Immanuel Kant"* in D_1 refers to the same real-world entity as the respective `customer` record in D_2 and `contact` record in D_3. Such *traceability links* a.k.a. *inter-model relationships* are commonly used in practice, e.g. [10–12,7]. In Fig. 1 we employ "tentacles" (⋗•⋅) to visualise the following: (1) The sameness of the records for *"Immanuel Kant"* in all three systems, (2) the sameness of the records for *"Arthur Schopenhauer"* in D_1 and D_2, (3) the dependency of a `case` in D_2 towards an existing `contract` D_1, including a witness (4) for the relationship between the `owner`-link in D_1 and the `foreign key` in D_2. We refer to these elements as *commonalities*.

$D_{1/2/3}$ may abstractly be formalised as graphs, see Sec. 2.1 for further details. Likewise, the collection of all commonalities are collected into a graph called D_0. Elements of D_0 witness related elements among $D_{1/2/3}$ and they must also respect node-edge-incidences (see witness (4)), such that their respective "tentacle"-ends (-•) are in fact graph morphisms $d_j : D_0 \to D_j$, $j \in I = \{1, 2, 3\}$.

For $|I| = 2$ the underlying star-shape of comprehensive systems (finite collections of arrows $(d_j)_{j\in\{1,\dots,n\}}$ with common source) reduces to the span shape $\bullet \leftarrow \bullet \rightarrow \bullet$, which is the underlying setting for triple graphs [4], the common source in the middle specifying the commonality graph. An extension of triple graphs are graph diagrams [5,6], a framework for *multi-ary* model synchronisation. Since multi-ary commonality relations such as the ternary tentacles of identical `client/customer/contact` records in Fig. 1 can not be encoded with multiple binary relations [13], one must distinguish relations of different arity in the underlying shape for graph diagrams: E.g., in Fig. 1, a shape that both supports binary (*dependency* between `contract/case` records) and ternary (*sameness* among `client/customer/contact` records) relations is needed. In larger system landscapes ($n > 3$), there may be many more commonality relations of arbitrary arity $k \le n$, which would cause a considerable amount of heterogeneity in the underlying shape for graph diagrams. Moreover, the graph diagram schema and hence the basic setting for implementations must be altered, each time new commonality relations are added.

We showed in [3,14] that comprehensive systems are a homogeneous generalisation of graph diagrams. They are *homogeneous*, because we need only one fixed shape to encode commonality relations of arbitrary arity (i.e. a star-shape with commonality graph D_0 as centre surrounded by graphs D_j representing related systems) and must not alter the base setting, if new relations are added. It is a *generalisation*, because we can implement each graph diagram as a comprehensive system, i.e. by jointly collecting different commonalities into D_0.

An important distinction, however, is that graph morphisms $d_j : D_0 \to D_j$ in comprehensive systems are allowed to be *partial*. For example $d_3 : D_0 \to D_3$ in Fig. 1 is undefined on (2), (3) and (4) in D_0: "*Arthur Schopenhauer*" is not represented in D_3 and `contract/case` records have no counterpart in D_3. We show that comprehensive systems with total homomorphisms form a category $\mathbb{CS}$ and, when restricting its morphisms to those morphisms that reflect definedness (yielding subcategory $\mathbb{RCS} \subseteq \mathbb{CS}$), SPO rewriting is possible. For this, we will consider the category $\mathtt{Par}(\mathbb{RCS})$, i.e. $\mathbb{RCS}$ equipped with *partial* morphisms, cf. Sect. 2.2. Although this requires handling both *intrinsic* and *extrinsic* partiality,[1] we can prove existence of all pushouts in this category (Theo. 1 in Sect. 4) and demonstrate applicability of SPO rewriting.

About this version. The present paper is an extended version of the paper [15] published in the proceedings of the 2020 edition of the *International Conference on Graph Transformation (ICGT 2020)* [16]. Compared to the conference version, we exchanged its rather simplistic running example with a more interesting scenario, which allows us to talk about heterogeneously typed models and typing-morphisms between Comprehensive Systems (Sect. 3.4). Furthermore, we added some more background material on graph rewriting in general, span and partial map categories, as well as upper adjoints (Sect. 2). Finally, we could transfer an important graph-based criterion for "*conflict-freeness*" for matches of SPO-rules to comprehensive systems (Theo. 2 in Sect. 5), which marks the most substantial extension compared to the conference version.[2]

2. Background

We expect the reader to have basic knowledge of category theory. For categorical artefacts, we employ the following notations: *Categories* like $\mathbb{C}$ will be denoted in a double-struck font. When distinguishing between members of $\mathbb{C}$, we write $|\mathbb{C}|$ (or just $\mathbb{C}$) for its objects and $\mathbb{C}^{\to}$ for its morphisms. $\mathbb{C}^{op}$ denotes the opposite category of $\mathbb{C}$, i.e. the category with the same objects as $\mathbb{C}$ but reversed arrows.

In general categories $\mathbb{C}$, there are identities $id_A : A \to A$ and composition $g \circ f$ for $f : A \to B$ and $g : B \to C$. $\mathbb{SET}$ is the category of sets and total mappings and we reserve the variable name $\mathbb{G}$ for categories that are based on a signature with unary operation symbols only, cf. Sect. 2.1. Monomorphisms ($\rightarrowtail$), epimorphisms ($\twoheadrightarrow$) and – if applicable – inclusions ($\hookrightarrow$) are highlighted by a special arrow notation. We furthermore expect the reader to be familiar with basic *universal constructions* like pullbacks, coproducts, and pushouts and their respective universal properties. We demonstrate our notations and terminology exemplified for pullbacks:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle g'}\uparrow & \lrcorner & \uparrow{\scriptstyle g} \\ D & \xrightarrow[f']{} & C \end{array} \tag{1}$$

(1) depicts a pullback diagram in $\mathbb{C}$, i.e. $A, B, C, D \in |\mathbb{C}|$ and $f, g, f', g' \in \mathbb{C}^{\to}$. When referring to a pullback in the text, we say that the span $(g' : D \to A, f' : D \to C)$ is the pullback of the co-span $(f : A \to B, g : C \to B)$. Alternatively, we call the commutative square described by the equation $f \circ g' = g \circ f'$, a pullback square. When domains and codomains are clear from the context we just write (g', f'), (f, g), or (g', f, f', g) to refer to the respective span, co-span or square. Sometimes, we call the morphism g' the pullback of g along f and D the *apex* or *pullback object*. The latter is highlighted with a small adjacent right angle. Since pullbacks are unique only up to isomorphism, we always assume a fixed choice of pullbacks, whenever we speak of a concrete pullback (of a given co-span).

Finally, we sometimes use the categorical term *diagram*. Formally, a diagram $\mathcal{D}$ in $\mathbb{C}$ is a functor $\mathcal{D} : \mathbb{S} \to \mathbb{C}$ where $\mathbb{S}$ is a small category, the *shape* of the diagram. Diagrams can also be defined as graph morphisms $\mathcal{D} : S \to \mathbb{C}$ from some schema graph S to (the underlying graph of) $\mathbb{C}$. However, we have to use the (equivalent) functorial definition here, because certain properties of comprehensive systems are then easier to prove.

2.1. Graph-like structures = abstract syntax

Fig. 1 shows (an excerpt of) a model of the stored information in the three systems $D_{1/2/3}$ and thus abstracts away from technical details such as programming languages, encoding and network communication. Models in Software Engineering

[1] "Intrinsic" refers to partiality *within* an $\mathbb{RCS}$-object, whereas "extrinsic" describes partiality of the morphisms *outside* of these objects.
[2] We discuss conflict-freeness in terms of rewriting in *span categories*, another main research area of Michael Löwe.

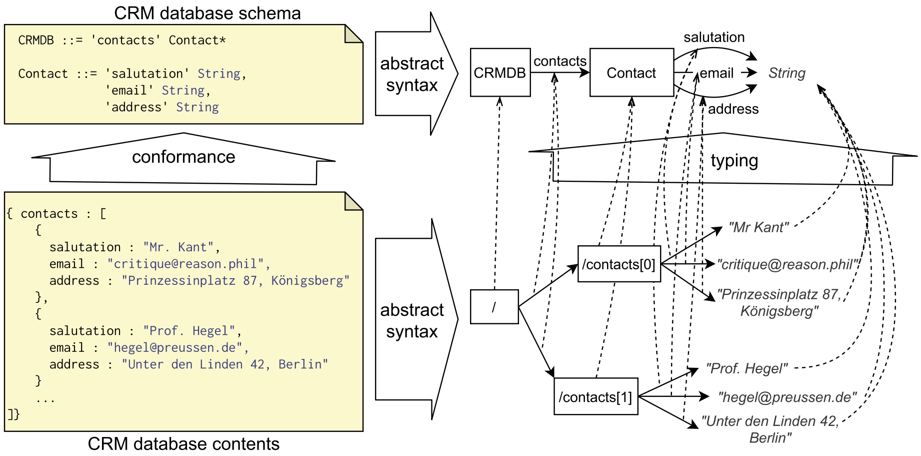


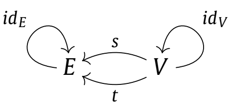
Fig. 2. Abstract syntax translation.

```
Σ_DG ::=
  sorts: V(ertices), (E)dges
  opns: s(ource), t(arget): E --> V
```

Listing 1: Signature Σ_{DG}.

(SE) come in all different kinds of concrete graphical or textual syntaxes. A strength of the graph transformation approach is its ability to treat such artefacts uniformly by applying a second abstraction step: When only considering the *abstract syntax*, i.e. "forgetting" the concrete syntax as illustrated in Fig. 2, every model can be considered as some sort of "graph-like structure". In Fig. 2, the concrete JSON syntax is translated into an *abstract syntax tree*, i.e. a directed acyclic graph, depicted on the bottom right. Moreover, the figure shows how a class of similarly-structured SE-models is constrained by typing. E.g. the members of D_3 should represent `contact` records with `salutation`, `email` and `address` fields. This may be specified by a schema or *metamodel* (top left in Fig. 2), which is translated to a graph as well. Hence, a well-known approach is to represent the conformance relation of a model to its metamodel as a graph (typing) homomorphism [17].

A category $\mathbb{G}$ is called *based on a signature with unary operation symbols only*, if it is isomorphic to a category of total algebras $\mathtt{Alg}(\Sigma)$ w.r.t. a signature Σ which only contains sorts and unary operation symbols. The simplest example in our context – and the rationale behind using letter $\mathbb{G}$ – are directed graphs, which is based on the signature Σ_{DG} in List. 1. This category $\mathtt{Alg}(\Sigma_{\mathrm{DG}})$ is isomorphic to a category of presheaves [18] $\mathbb{SET}^{\mathbb{B}^{op}}$ with $\mathbb{B} := \mathbb{B}_{\mathrm{DG}}$ being the small category shown in (2).



(2)

We do not endorse directed graphs in particular and could likewise choose $\mathbb{G}$ to be given by hypergraphs [19,20], bipartite artefacts like condition-event-nets [21], or E-Graphs [22], which distinguish between object and value nodes as well as reference and attribute edges and therefore are well-aligned with object diagrams in UML [23] and MOF [24]. It is well-known that all these categories are topoi and thus possess all limits (e.g. pullbacks) and colimits (coproducts, pushouts) [18].

Objects of such categories $\mathbb{G}$ have been called "graph-like structures" [25], and thus we will overload the term "graph" and call $\mathbb{G}$-objects "graphs" and $\mathbb{G}$-morphisms "graph (homo-)morphisms" bearing in mind the above mentioned more general setting. Furthermore, objects of $\mathbb{G}$ are sufficiently concrete, i.e. set-based, and we can talk about *elements*: A graph $G \in |\mathbb{G}|$ can be seen as a functor $G : \mathbb{B}^{op} \to \mathbb{SET}$. An element $x \in G$ of graph G is a member of the carrier set $G(s)$ for some sort $s \in |\mathbb{B}|$. Likewise, "$\forall x \in G$" means "for all x of any sort s in the carrier sets of G".

Remark. $\mathbb{G}$ will serve as the base category (or base structure) for assembling comprehensive systems. Actually, we could have traded $\mathbb{G}$ for more general (weak) adhesive (HLR) categories [26] w.r.t. an admissible subclass $\mathscr{M}$ of all monomorphisms.[3] Adhesive HLR categories have mainly been introduced to model graph transformations featuring calculations on *attributes*: Note e.g. that some nodes in Fig. 2 are using an italic font and have no node outline to highlight the fact that

[3] It seems that the subsequent proofs can still be carried out, if $\mathbb{G}$ is such a more general structure. We will provide respective facts from time to time.

(a) DPO derivation. (b) SPO derivation.

Fig. 3. Graph rewriting.

these nodes represent *values*, e.g. `Strings` and `Integers`. Attributed graphs [22] model this by associating an algebra with these base type nodes. The latter poses some challenges regarding adhesiveness, in turn requiring to work with a special subclasses of morphisms, which are isomorphic (static) on the "data part". However, we continue with graph-like structures because we are not focusing on attributes here and we want to stay in the tradition of Michael Löwe, who originally investigated graph-like structures only. To be technically sound, we assume that all graphs featured in forthcoming examples implicitly contain nodes for all possible base type values and that objects created by universal constructions do not "rename" them. Our conjecture is that the attribution concept introduced by attributed graphs can seamlessly be incorporated into comprehensive systems in the future.

2.2. SPO rewriting

Graph rewriting also known as *graph transformation* [27] is a powerful and well-investigated formal tool that generalises traditional string-grammar rewriting [28]. The common de-facto standard of *algebraic graph transformation* was introduced in 1973 by Ehrig et al. [29] in terms of the *double-pushout approach* (DPO): Consider Fig. 3a, a DPO-rule is given by a span (l, r) of morphisms in a "suitable" category $\mathbb{C}$. Traditionally this category $\mathbb{C}$ was given by the category of labelled directed graphs. It was later abstracted to a general setting of *High Level Replacement (HLR)* [30] structures and since the discovery of adhesive categories [31], the ambient category $\mathbb{C}$ is generally assumed to be an arbitrary (weak) adhesive (HLR) category [26]. Given a *match* m (i.e. morphism incident to l), one can rewrite G to H via rule (l, r) at match m, written $G \overset{(l,r)@m}{\rightsquigarrow} H$, if there exist morphisms $l', r', \overline{m}, m'$ such that the squares (a) and (b) are pushout squares. The morphism m' is called the *co-match*, object C the *context*, and span (l', r') the *trace* of the derivation. A rule is called *linear* if both l and r are monomorphisms (or members of a special class of monomorphisms $\mathscr{M}$ resp.), which is the most common case since it guarantees that both pushout squares are well-behaved. This well-behavedness (exactness) property refers to the so-called (weak) *van Kampen* property [31], which asserts a certain interplay between pushouts and pullbacks. In practice, a rule application at a given match involves constructing a pushout complement (a) followed by a pushout construction (b). Intuitively, the former models "deletion" while the latter models "insertions" when the rule is linear. Raoult [32] was the first to propose replacing this construction with a conceptually simpler variant, which only requires a single pushout when exchanging total with partial morphisms. The construction in [32] was set-based and could not model deletions properly. Following Raoult's idea, Kennaway [33] and Löwe [1], independently of each other, developed the theory for single pushout rewriting:

Definition 1 *(SPO - rule, match, derivation, conflict-freeness).* An SPO *rule* is a *partial* morphism $\rho : L \rightharpoonup R \in \mathtt{Par}(\mathbb{C})^{\rightarrow}$, i.e. a morphism of the category of partial morphisms $\mathtt{Par}(\mathbb{C})$ over a given category of total morphisms $\mathbb{C}$ (note the partial arrow-tips in Fig. 3b). A *match* for ρ at (host) $G \in \mathbb{C}$ is a *total* morphism $\mu : L \to G \in \mathbb{C}^{\rightarrow}$. A pushout of ρ and μ in $\mathtt{Par}(\mathbb{C})$ generates the (SPO-) *derivation* $G \overset{\rho@\mu}{\rightsquigarrow} H$ with *trace* ρ' and *co-match* μ', see Fig. 3b. The match μ is called *conflict-free*, if μ' is a total morphism.

The relation between DPO and SPO as well as other algebraic graph transformation approaches such as e.g. *sesqui pushout rewriting (SqPO)* [34] has been a recurring theme in Michael's work [35,36]. In fact, DPO with left-linear rules is a special case of SPO at *conflict-free* matches [25]. We discuss this relationship and a concrete characterisation of conflict-freeness for comprehensive systems in Sect. 5.

First, we have to check whether the ambient category actually admits SPO rewriting, i.e. does the category of partial morphisms $\mathtt{Par}(\mathbb{C})$ over $\mathbb{C}$ possess pushouts? A significant contribution to answer this question is the work of Hayman and Heindel [37], which provides a sufficient and necessary condition in terms of *hereditary pushouts* and *upper adjoints*. We will explain both concepts in the following subsections.

2.3. Span and partial map categories

Michael had the courage to leave the comfortable world of *total* morphisms and utilised *partial* morphisms [38] for the SPO approach. While other researchers adhered to total morphisms, he forcefully followed through with extrinsic partiality

(a) Span relation. (b) Span composition.

Fig. 4. Arrows and composition in $\mathtt{Par}(\mathbb{C})$.

and proved that it is worthwhile [1]. Originally, he used the following definition[4]:

A partial morphism $f : A \rightharpoonup B$ is defined by a pair $(dom(f), f)$ where $dom(f) \subseteq A$ is a subobject of A and $f : dom(f) \to B$ is a total morphism. This notion is equivalent with the common approach to consider the category of partial morphisms $\mathtt{Par}(\mathbb{C})$ over a category with total morphisms $\mathbb{C}$ as a subcategory of the span category $\mathtt{Span}(\mathbb{C})$, for a more comprehensive survey we refer to the seminal work of Robinson and Rosolini [38].

We recall the notion of span categories here: Let $\mathbb{C}$ be an arbitrary category, which has all pullbacks. A *concrete span* between A and B ($A, B \in |\mathbb{C}|$) is given by a pair of morphisms $(m : X \to A, f : X \to B)$ sharing the same domain. Among all concrete spans between A and B we can define a relation $\approx_{A,B}$ that is defined as follows: Two spans between $(m : X \to A, f : X \to B)$ and $(m' : X' \to A, f' : X' \to B)$ between the same pair of objects (A, B) are said to be in relation $\approx_{A,B}$ if and only if there exists an isomorphism $i : X \to X'$ such that $m = m' \circ i$ and $f = f' \circ i$, cf. Fig. 4a. Due to the isomorphism property, these relations form a family of equivalence relations $\approx := (\approx_{A,B})_{A,B \in |\mathbb{C}|}$. A representative (m, f) of a class of equivalent spans is called an abstract span and denoted by $[m, f\rangle$. The span category $\mathtt{Span}(\mathbb{C})$ over $\mathbb{C}$ shares the same class of objects with $\mathbb{C}$, i.e. $|\mathtt{Span}(\mathbb{C})| = |\mathbb{C}|$, and the hom-set $\mathtt{Span}(\mathbb{C})(A, B)$ of morphisms between a pair of objects $A, B \in |\mathtt{Par}(\mathbb{C})|$ is given by the set of all abstract spans between A and B. Identities are given by abstract spans that have pairs of $\mathbb{C}$-identities as representatives, while composition is defined via pullback, see Fig. 4b: The composition $[n, g\rangle \circ [m, f\rangle$ of abstract spans $[m, f\rangle$ and $[n, g\rangle$ is given by the abstract span $[m \circ n', g \circ f'\rangle$ where n' and f' are derived as the pullback of (f, n).

The underlying category $\mathbb{C}$ can be embedded into $\mathtt{Span}(\mathbb{C})$. The functor that embeds $\mathbb{C}$ into $\mathtt{Span}(\mathbb{C})$ is called the *graphing functor* Γ [39] due to historic reasons.[5] This functor is an identity-on-objects functor and maps every morphism to a trivial span:

$$\Gamma : \begin{cases} \mathbb{C} \to \mathtt{Span}(\mathbb{C}) \\ f \in \mathbb{C}(A, B) \mapsto [id_A, f\rangle \in \mathtt{Span}(\mathbb{C})(A, B) \end{cases}$$

The span category $\mathtt{Span}(\mathbb{C})$ has some notable subcategories. We are most interested in the category of partial morphisms $\mathtt{Par}(\mathbb{C}) \subseteq \mathtt{Span}(\mathbb{C})$, which imposes the restriction that the left legs m of all $\mathtt{Par}(\mathbb{C})$-morphisms $[m, f\rangle$ must be monomorphisms. In the following, we will solely focus on the span category $\mathtt{Par}(\mathbb{C})$ and denote arrows in this category $[m, f\rangle : A \rightharpoonup B$ with a partial arrow-tip to distinguish them from total arrows $f : A \to B \in \mathbb{C}^{\to}$ and also use Γ to denote the embedding functor from $\mathbb{C}$ into $\mathtt{Par}(\mathbb{C})$. When the inner leg m of a partial arrow span $[m, f\rangle$ is an isomorphism, the class $[m, f\rangle$ is the same as $[id_A, f\rangle$, hence $[m, f\rangle$ has a pre-image under Γ and can be seen as a *total* morphism. In this case, we denote it with a regular arrow-tip: $[m, f\rangle : A \to B$.

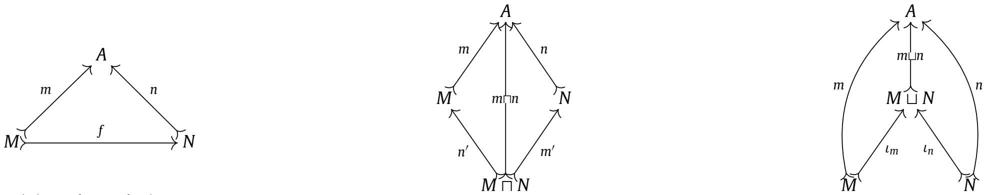
For $\mathbb{C} := \mathbb{G}$, we can refer to elements inside objects of $\mathbb{G}$. Hence, there are special monomorphisms: *inclusion*-arrows $m : A \hookrightarrow B$ – highlighted by a hook-arrow – that do not "rename" elements, i.e. with $\mathbb{B}$ being the underlying signature of $\mathbb{G}$, for all $s \in |\mathbb{B}|$ there are inclusions $A(s) \subseteq B(s)$ between carrier sets. Since we are working with classes $[m, f\rangle$ of spans in $\mathtt{Par}(\mathbb{G})$, we might as well choose m as an inclusion as the chosen representative for the abstract span. In this case, we overload the morphism names of right legs, i.e. a partial morphism $f : A \rightharpoonup B \in \mathtt{Par}(\mathbb{G})$ is the class $[\subseteq, f\rangle$, which is represented by a concrete span $(\subseteq : dom(f) \hookrightarrow A, f : dom(f) \to B)$, which brings us back to Michaels original definition of partial morphisms in [25]. Such a partial morphism f is "total", if the inclusion is an identity. We will use $\mathbb{G}$-inclusions whenever there is a choice for monomorphisms (replacing $\rightarrowtail$ with $\hookrightarrow$).

Finally, we can introduce the notion of *hereditary pushouts*, one requirement for the existence of pushouts in $\mathtt{Par}(\mathbb{C})$.

Definition 2 *(Hereditary pushout [33,40]).* A pushout in $\mathbb{C}$ is called *hereditary*, if its Γ-image is a pushout in $\mathtt{Par}(\mathbb{C})$. If all pushouts exist in $\mathbb{C}$ and they are all hereditary, we say that $\mathbb{C}$ is a *hereditary pushout category*.

[4] ... which is also the way to define "intrinsic" partiality, i.e. partial operations of partial algebras. The combination of both is very thoroughly investigated in [2].

[5] Recall that every function $f : A \to B$ gives rise to a relation $graph(f) = \{(a, f(a)) \mid a \in A\} \subseteq A \times B$, called the *graph* of the function. The function $f : A \to B$ thus turns into the span (id_A, f) of projections from the graph to the domain and codomain of f.



(a) Order relation. (b) Meets. (c) Joins.

Fig. 5. Subobject lattices.

The following result can be found in [33]:

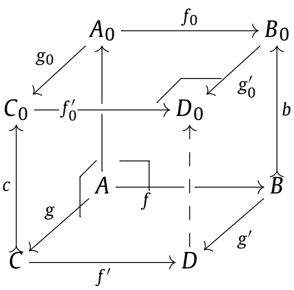
Proposition 1. *If the* Γ*-image of a* $\mathbb{C}$*-span has a pushout in* $\mathtt{Par}(\mathbb{C})$*, then this cocone consists of two total morphisms, which are the* Γ*-image of the pushout of this span in* $\mathbb{C}$*.* □

The following result was stated in [37] but fully worked out already in [1]:

Proposition 2. $\mathbb{G}$ *is a hereditary pushout category.* □

Finally, hereditariness can equivalently be characterised by a condition which is similar to the so-called weak vertical *Van Kampen* property [41,40,39]:

Proposition 3 *(Equivalent characterisation of hereditariness [40]).*



(3)

A pushout $g_0' \circ f_0 = f_0' \circ g_0$ *is* hereditary, *if and only if in any commutative cube as in* (3) *with rear faces being pullbacks and vertical front left and back right arrows (c and b in* (3)*) being monomorphisms, the following equivalence holds: The bottom face is a pushout if and only if (1) the two front faces are pullbacks and (2) the vertical front right arrow (the dashed arrow in* (3)*) is a monomorphism.* □

2.4. Subobject lattices and upper adjoints

The following notions of sub-object lattices are well-known from the literature, e.g. [18]: Let $\mathbb{C}$ be a category with pullbacks. For every object A, there is a pre-order $(Sub(A), \sqsubseteq)$ of (abstract) A-subobjects. An A-subobject $[m] \in Sub(A)$ is given by an equivalence class $[m] := \{m' \mid m \equiv_A m'\}$ of monomorphisms $m' : M' \rightarrowtail A$ with codomain A. The respective equivalence relation $\equiv_A$ is defined on pairs of monomorphisms $m : M \rightarrowtail A$ and $m' : M' \rightarrowtail A$: $m \equiv_A m'$, if and only if there exists an isomorphism $i : M \to M'$ such that $m = m' \circ i$ holds.

In the sequel, we will use monomorphisms $m, n, \ldots$ as representatives for subobjects $[m], [n], \ldots$ (i.e. seamlessly adding and removing brackets on demand) and implicitly assume their domain to be the corresponding upper case letter $M, N, \ldots$. Note that when $\mathbb{C} = \mathbb{G}$, we can again choose representatives of subobjects to be inclusions since we are working with classes of monomorphisms.

The order relation $\sqsubseteq$ between two subobjects $[m]$ and $[n]$, written $[m] \sqsubseteq [n]$, holds, if there is a (necessarily monic and unique) $\mathbb{C}$-morphism $f : M \to M'$ with $m = n \circ f$, see Fig. 5a. Furthermore, the existence of pullbacks in $\mathbb{C}$ turns the partial order $\sqsubseteq$ into a semi-lattice, where the meet $[m] \sqcap [n]$ (think intersection) of two subobjects $[m]$ and $[n]$ is given by the diagonal of the pullback of m and n (recall that pullbacks always preserve monomorphisms), cf. Fig. 5b. The existence of joins $[m] \sqcup [n]$ (i.e. effective unions), see Fig. 5c, is not always given in an arbitrary category $\mathbb{C}$ with pullbacks. However, if $\mathbb{C}$ has coproducts and images that are stable under pullback, the subobject semi-lattice $Sub(A)$ has joins, hence, becomes a full lattice:

Definition 3 *(Images and effective unions).* A category $\mathbb{C}$ is said to have *images* if and only if for every $\mathbb{C}$-morphism $f : A \to B$ there is a least (w.r.t. $\sqsubseteq$) $[m] \in Sub(B)$ such that $f = m \circ e$ for some $e : A \twoheadrightarrow M$ and we call the domain of m the *image object* $Im(f) := M$. Further, if $\mathbb{C}$ has coproducts, then, for a family $([m_x])_{x\in X}$ of subobjects of A for some index set X, we define the effective union as $\bigsqcup_{x\in X}[m_x] := [j : Im(u) \rightarrowtail A]$ where u is the unique mediator morphism $u : \coprod_{x\in X} M_x \to A$ w.r.t. the family $(m_x : M_x \rightarrowtail A)_{x\in X}$ and j arises from the image factorisation $u = j \circ e$ of u. Thus, in particular, for all $y \in X$

$$[m_y] \sqsubseteq \bigsqcup_{x\in X}[m_x]. \tag{4}$$

Now we can discuss upper adjoints w.r.t. to inverse image functions, the second ingredient for the existence of pushouts in $\mathtt{Par}(\mathbb{C})$. This notion can be described as a special case of the pullback functor and its right adjoint: Recall, that every pre-order is a category where the hom-set between two objects has at most one element. Thus, partial orders are skeletal categories and monotone functions[6] are functors between such categories.

Definition 4 *(Inverse images and upper adjoints [37]).* Let $f : A \to B$ be given in a category $\mathbb{C}$ with pullbacks. We denote by $f^{-1} : Sub(B) \to Sub(A)$ the inverse image function which assigns to $[m] \in Sub(B)$ its pullback along f. A monotone function $\forall_f : Sub(A) \to Sub(B)$ is called an *upper adjoint* of f^{-1}, if for all $[n] \in Sub(A)$ and $[m] \in Sub(B)$:

$$f^{-1}[m] \sqsubseteq [n] \iff [m] \sqsubseteq \forall_f[n] \tag{5}$$

Note that $\forall_f$ is unique, if it exists [37], and that f^{-1} is monotone, since pulling back is functorial and preserves monomorphisms. We remark that, in $\mathbb{G}$, the upper adjoint is the right-adjoint of the pullback functor f^{-1}. In [42], it was shown that for $[n] \in Sub(A)$ validity of the condition

$$\forall x, y \in A : f(x) = f(y) \Rightarrow (x \in N \iff y \in N) \tag{6}$$

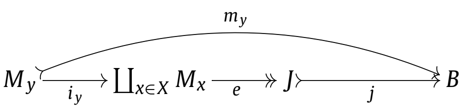
implies $\forall_f[n] : (f \circ n)(N) \hookrightarrow B$ and that the co-unit $\varepsilon_n : f^{-1}(\forall_f n) \to n$ is an isomorphism.

There is a generic construction for upper adjoints in categories with effective unions:

Proposition 4 *(Upper adjoints). Let $\mathbb{C}$ be a category with pullbacks, coproducts and images that are both stable under pullback. Let further $f : A \to B \in \mathbb{C}^{\to}$ and $[n] \in Sub(A)$, then $\forall_f[n] := \bigsqcup\{[m] \in Sub(B) \mid f^{-1}[m] \sqsubseteq [n]\}$ is the upper adjoint of f^{-1}.*

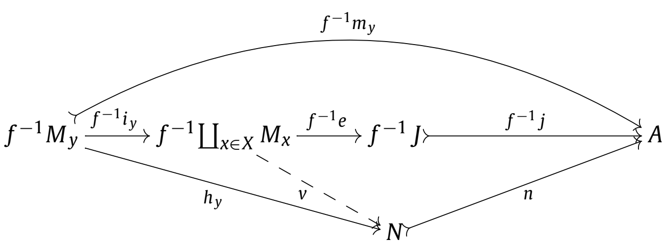
Proof. To prove that $\forall_f$ is monotone, assume $[n], [n'] \in Sub(A)$ with $[n] \sqsubseteq [n']$. Hence $X := \{[m] \in Sub(B) \mid f^{-1}[m] \sqsubseteq [n]\} \subseteq \{[m] \in Sub(B) \mid f^{-1}[m] \sqsubseteq [n']\} =: X'$ and thus there is the mediator $u : \coprod_{[m]\in X} M \to \coprod_{[m]\in X'} M$, such that $\forall_f[n']$ becomes a factor in a decomposition of $\coprod_{[m]\in X} M \to A$. Since $\forall_f[n]$ is the least of these factors, we obtain $\forall_f[n] \sqsubseteq \forall_f[n']$.

In equivalence (5) "$\Rightarrow$" follows immediately from (4), such that it remains to prove "$\Leftarrow$". For this it is sufficient to show $f^{-1}(\forall_f[n]) \sqsubseteq [n]$ for all $[n] \in Sub(A)$, because f^{-1} is monotone.[7] Let $[n] \in Sub(A)$ be arbitrary and $\forall_f[n] := \bigsqcup_{x\in X} m_x : J \rightarrowtail B$. Further, let $j : J \rightarrowtail B$ be the representative of $\bigsqcup_{x\in X}[m_x]$ and fix some $y \in X$. Then there is the coproduct injection $i_y : M_y \to \coprod_{x\in X} M_x$ and by the definition of $\bigsqcup_{x\in X}[m_x]$ in Definition 3, we obtain the diagram



(7)

which is mapped by f^{-1} (interpreted as pullback functor) to the upper part of the following diagram:



[6] A function $U : (X, \leq_X) \to (Y, \leq_Y)$ between two partially ordered sets is called monotone, if it preserves the order, i.e. $\forall x, x' \in X : x \leq_X x' \Rightarrow U(x) \leq_Y U(x')$.

[7] If pullback functors have right-adjoints, this inequality corresponds to the co-unit of adjunction $f^{-1} \dashv \forall_f : \mathbb{C} \downarrow A \to \mathbb{C} \downarrow B$.

In this diagram, h_y exists with $n \circ h_y = f^{-1}m_y$, because $y \in X$ and thus $f^{-1}[m_y] \sqsubseteq [n]$ by the definition of $\forall_f$. Because we assumed coproducts to be stable under pullback, the image $f^{-1}\coprod_{x\in X} M_x$ of $\coprod_{x\in X} M_x$ under f^{-1} is the coproduct of $(f^{-1}M_x)_{x\in X}$ and $f^{-1}i_y$ is the respective coproduct injection. We obtain v as the unique mediator out of this coproduct w.r.t. all arrows $\{h_y \mid y \in X\}$, i.e. $v \circ f^{-1}i_y = h_y$ and hence for all $y \in X$: $n \circ v \circ f^{-1}i_y = f^{-1}m_y = f^{-1}j \circ f^{-1}e \circ f^{-1}i_y$, the last equality by functoriality of f^{-1}. By universality of coproducts this yields $n \circ v = f^{-1}j \circ f^{-1}e$. Since pullbacks preserve images, the latter term in the above equation is the image factorisation of $n \circ v$ and hence $f^{-1}\forall_f[n] = f^{-1}(\bigsqcup_{x\in X}[m_x]) \sqsubseteq [n]$, the former being the least, the latter being some subobject of A factoring through $n \circ v$. □

3. Comprehensive systems

For the rest of the paper, we fix a sufficiently large natural number n (usually a number which always exceeds the possible number of distributed systems under consideration). Hence, all constructions are parametrised by the constant n.

3.1. Definitions and background

Definition 5 *(Comprehensive system).* Let $(C_i)_{0\le i\le n}$ be an $n+1$-tuple of $\mathbb{G}$-objects. We call

- $(C_j)_{1\le j\le n}$ the *Components* and
- C_0 the (graph of) *Commonality Representatives*

of a *Comprehensive System*

$$\mathbf{C} := (c_j : C_0 \rightharpoonup C_j)_{1\le j\le n}$$

i.e. an n-tuple of partial graph morphisms $(c_j)_{1\le j\le n}$, which we call *projections*.[8]

In order to make reading easier, we always use letter i, if indexing comprises graphs $C_0, C_1, \dots, C_n$ and we use letter j, if indexing is only over the components $C_1, \dots, C_n$. Moreover, we denote the whole comprehensive system with a bold face letter $\mathbf{C}$.

Comprehensive systems admit an all-embracing view on a system of possibly heterogeneously typed components, in which all inter-model relationships are coded, cf. Fig. 1. They have been treated on the level of graphs in [8] and – on a more abstract level – in [43].

Definition 6 *(Morphism of comprehensive systems).* Let $\mathbf{C} := (c_j : C_0 \rightharpoonup C_j)_{1\le j\le n}$ and $\mathbf{D} := (d_j : D_0 \rightharpoonup D_j)_{1\le j\le n}$ be two comprehensive systems. A morphism $\mathbf{f} : \mathbf{C} \to \mathbf{D}$ is a family $(f_i : C_i \to D_i)_{0\le i\le n}$ of total $\mathbb{G}$-morphisms, such that for all $1 \le j \le n$ and all $x \in C_0$:

$$c_j(x) \text{ is defined} \implies d_j(f_0(x)) \text{ is defined} \tag{8}$$

and

$$(d_j \circ f_0)(x) = (f_j \circ c_j)(x). \tag{9}$$

Whenever we mention morphisms $\mathbf{f} : \mathbf{C} \to \mathbf{D}$ between comprehensive systems, we implicitly assume the components of $\mathbf{C}$ and $\mathbf{D}$ be denoted as in Definition 5 and we assume the constituents of $\mathbf{f}$ be denoted as in Definition 6.

There is the obvious identical morphism $\mathbf{id_C}$ for each comprehensive system $\mathbf{C}$ and composition can be defined componentwise. Hence we obtain

Proposition 5 *(Category $\mathbb{CS}$ and component functors). Let $\mathbb{G}$ be a category as described above and let n be given as above.*

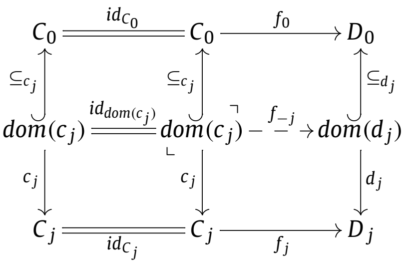
- *Comprehensive Systems and morphisms between them constitute a category, denoted $\mathbb{CS}_{n,\mathbb{G}}$. Since n is a constant, we omit index n and we also write just $\mathbb{CS}$, if the base category is clear from the context.*
- *For each $i \in \{0, \dots, n\}$ there is the component functor $(\_)_i : \mathbb{CS} \to \mathbb{G}$ defined by $(\_)_i(\mathbf{f} : \mathbf{C} \to \mathbf{D}) = f_i : C_i \to D_i$ for any $\mathbf{f}$.*

Let us investigate the consequences of strengthening the definition of $\mathbb{CS}$-morphisms (Definition 6) by additionally requiring

$$d_j \circ [id_{C_0}, f_0\rangle = [id_{C_j}, f_j\rangle \circ c_j \tag{10}$$

[8] One might consider elements of C_0 to be tuples (of arbitrary size $k \le n$), in which common elements are listed, hence the term "projection" for the c_j.

to be a commutative square in $\mathtt{Par}(\mathbb{G})$. Recall that the definition of composition of partial morphisms involves pullbacks and consider the $\mathbb{G}$-diagram in (11):



(11)

The bottom left and top right squares are pullbacks, which are needed for the composition of $[id, f_j\rangle \circ c_j$ and $d_j \circ [id, f_0\rangle$ respectively, cf. Fig. 4b. Condition (10) requires the apexes of these pullbacks to be equal and since the bottom left pullback is constructed along an identity, this apex can be chosen as $dom(c_j)$. Hence, we may define a morphism in comprehensive systems as a family $(f_{-n}, \dots, f_{-j}, \dots, f_0, \dots, f_j, \dots, f_n)$ of $\mathbb{G}$-morphisms defined on domains of definitions of projection morphisms $dom(c_j)$, on C_0, as well as on components C_j, as shown in the right-hand side of (11), such that the upper squares $f_0 \circ \subseteq_{c_j} = \subseteq_{d_j} \circ f_{-j}$ are pullbacks and the lower squares $f_j \circ c_j = d_j \circ f_{-j}$ commute. Translating this notion back into the formulation of Definition 6, the upper pullback turns the implication in (8) into an equivalence (while commutativity of the lower right square corresponds to (9)). In such a way, definedness of a projection is not only preserved but also reflected:

Definition 7 *(Reflective $\mathbb{CS}$-morphism).* A morphism $\mathbf{f} : \mathbf{C} \to \mathbf{D}$ (Definition 6) is called *reflective* if and only if the implication (8) is an equivalence.[9]

Further, let $\mathbb{RCS}$ denote the category of comprehensive systems ($|\mathbb{RCS}| = |\mathbb{CS}|$) and reflective $\mathbb{CS}$-morphisms ($\mathbb{RCS}^{\to} \subsetneq \mathbb{CS}^{\to}$) between them.

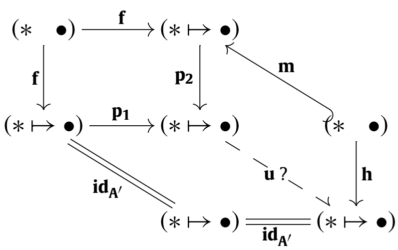
Let us investigate why we have to restrict ourself to this sub-category for applying SPO rewriting on comprehensive systems.

3.2. Why must definedness be reflected?

Since our goal is to show that SPO rewriting is applicable for comprehensive systems, we must show that the respective category of partial morphisms has all pushouts. Assume we would look for the existence of pushouts in $\mathtt{Par}(\mathbb{CS})$. In this case let's consider for $n = 1$ two simple comprehensive systems:

Counterexample 1 *(Pushouts in $\mathtt{Par}(\mathbb{CS})$).* Let $\mathbb{G} = \mathbb{SET}$ and $A_0 = \{*\}$ and $A_1 = \{\bullet\}$ be two one-element sets and let $\mathbf{A} = (a_1 : A_0 \rightharpoonup A_1)$ with a_1 the totally undefined map depicted with $(* \quad \bullet)$ and $\mathbf{A}' = (a_1{}' : A_0 \to A_1)$ with $a_1{}'$ the unique total map from A_0 to A_1 depicted $(* \mapsto \bullet)$. If we only work with preservation of definedness, then morphism $[\mathbf{id_A}, \mathbf{f}\rangle : \mathbf{A} \to \mathbf{A}'$, in which f_0 maps $*$ to $*$ and f_1 maps $\bullet$ to $\bullet$, is an admissible morphism. We claim that the span $\mathbf{A}' \xleftarrow{[\mathbf{id_A}, \mathbf{f}\rangle} \mathbf{A} \xrightarrow{[\mathbf{id_A}, \mathbf{f}\rangle} \mathbf{A}'$ does not possess a pushout in $\mathtt{Par}(\mathbb{CS})$.

If there would be a pushout of this span of two total morphisms in $\mathtt{Par}(\mathbb{CS})$, then, by Proposition 1, it must coincide with the pushout of them in $\mathbb{CS}$. Since $\mathbf{f}$ is an epimorphism in $\mathbb{CS}$ (because all f_i are epimorphims in $\mathbb{G}$), the pushout in $\mathbb{CS}$ must have $\mathbf{p_1} = \mathbf{p_2} = \mathbf{id_{A'}}$ as cocone, see the left top square in (12).



(12)

The two partial morphisms $[\mathbf{m}, \mathbf{h}\rangle$ and $[\mathbf{id_{A'}}, \mathbf{id_{A'}}\rangle$ let the outer rectangle of partial morphisms commute, i.e.

$$[\mathbf{m}, \mathbf{h}\rangle \circ [\mathbf{id_A}, \mathbf{f}\rangle = [\mathbf{id_{A'}}, \mathbf{id_{A'}}\rangle \circ [\mathbf{id_A}, \mathbf{f}\rangle$$

[9] In categories of partial algebras (and total homomorphisms between them), this definition coincides with the subclass of *closed* homomorphisms. See [2] for a thorough investigation of closed, full, and normal homomorphisms.

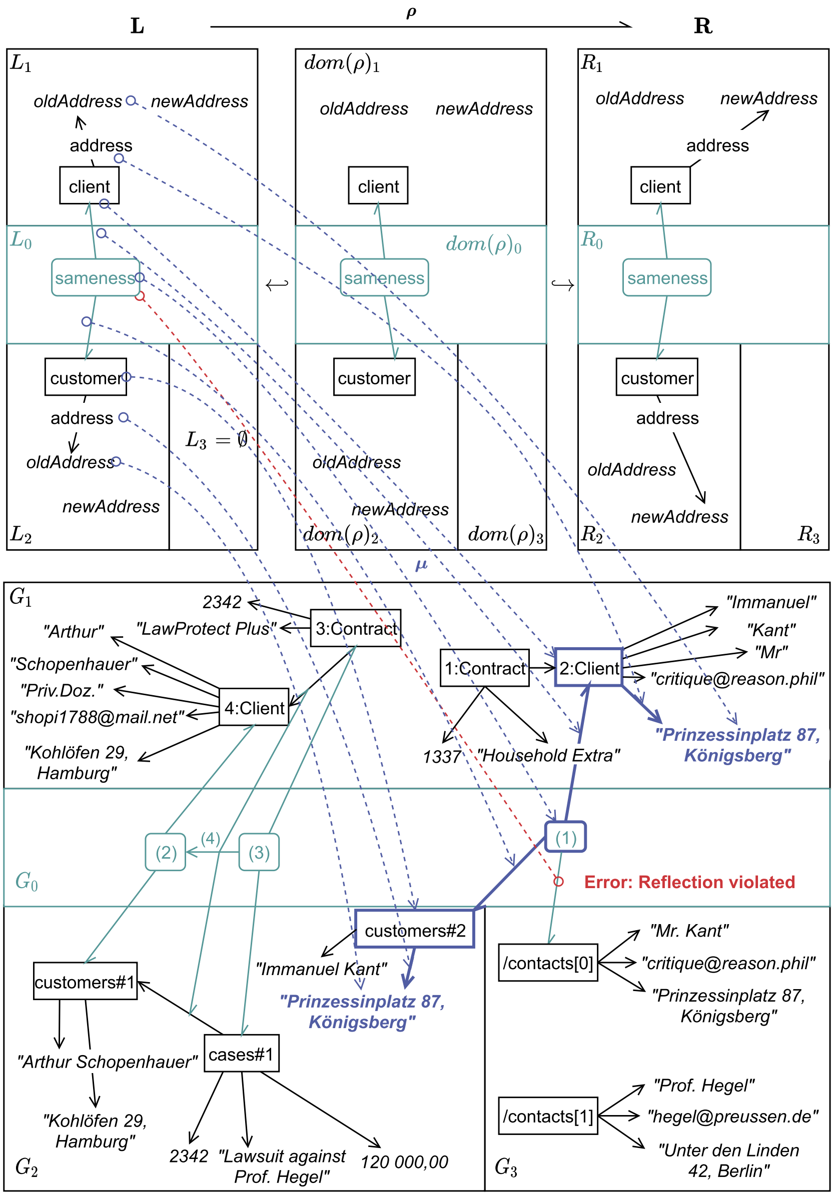


Fig. 6. Matching with reflective $\mathbb{CS}$-morphisms.

in $\mathtt{Par}(\mathbb{CS})$, because the pullback object of $\mathbf{m}$ and $\mathbf{f}$ equals the pullback object of $\mathbf{id}_{\mathbf{A}'}$ and $\mathbf{f}$ in $\mathbb{CS}$. If there would be a unique mediator $\mathbf{u}$, see the dashed line in the diagram, we must have $\mathbf{u} = [\mathbf{id}_{\mathbf{A}'}, \mathbf{id}_{\mathbf{A}'}\rangle$, because the lower rhombus must be commutative. However, for this $\mathbf{u}$ the right rhombus fails to be commutative, because $\mathbf{u} \circ [\mathbf{id}_{\mathbf{A}'}, \mathbf{p_2}\rangle = [\mathbf{id}_{\mathbf{A}'}, \mathbf{id}_{\mathbf{A}'}\rangle \neq [\mathbf{m}, \mathbf{h}\rangle$.

This example shows that we cannot expect to have all pushouts in $\mathtt{Par}(\mathbb{CS})$, even if the two morphisms, for which the pushout shall be constructed, are total monomorphisms. Intuitively, we can expect to have pushouts (of monomorphisms), i.e. unions, only if there are unique embeddings. The example of the two different "embeddings" $[\mathbf{m}, \mathbf{h}\rangle$ and $[\mathbf{id}_{\mathbf{A}'}, \mathbf{id}_{\mathbf{A}'}\rangle$ shows that this is not the case in $\mathtt{Par}(\mathbb{CS})$. We will show in the sequel that working in the category $\mathbb{RCS}$ removes these deficits.

Finally, we give an example why the restriction to the category $\mathbb{RCS}$ is useful in practice, too. The top part of Fig. 6 shows a rule $\boldsymbol{\rho} : \mathbf{L} \rightharpoonup \mathbf{R}$ that updates the `address`-field of `customer` and `client` records representing the same person in CoM and CaM. This rule, however, neglects the possibility of `contact` records in the CRM. The reflection-property prevents application of an underspecified rule at match $\boldsymbol{\mu}$ in a host comprehensive system $\mathbf{G}$: E.g. with $\mathbf{G} = \mathbf{D}$ ($\mathbf{D}$ known from Fig. 1), $\boldsymbol{\rho}$ cannot be applied on the records for *"Immanuel Kant"*, see the lower half of Fig. 6: The definedness of projection g_3 on the commonality witness (1) is not reflected at the node `sameness` in $\mathbf{L}$. In this way reflective rules restrict the application of rules that are underspecified (not taking all system components into consideration). Thus, the reflection requirement can be seen as a kind of built-in *negative application condition* [44]. In [15] we also demonstrated that this requirement serves to prevent multiple applications of rules which involve commonalities.

3.3. Important properties

In the sequel, we will use formulations like "a property is valid *componentwise*" in $\mathbb{RCS}$ or some construction "is carried out *componentwise*". Since many of the following considerations are based on this methodology, we give a formalisation: "Pushout", "Pullback", "Monomorphism", "Commutativity" impose truth of a predicate (a certain property) on a diagram in a category $\mathbb{C}$. For pushouts and pullbacks the underlying diagram is a square, for the predicate "Monomorphism" it is a single arrow, for "Commutativity" it is an appropriate triangle of arrows. E.g. $\mathbb{RCS}$-morphism $\mathbf{f}: \mathbf{A} \to \mathbf{B}$ is a componentwise monomorphism means that each f_i is a $\mathbb{G}$-monomorphism. More precisely: Given a diagram $\mathcal{D}: \mathbb{S} \to \mathbb{RCS}$ of any of the above mentioned shapes in $\mathbb{RCS}$, let $\mathcal{D}_i := (\_)_i \circ \mathcal{D}: \mathbb{S} \to \mathbb{G}$ with component functor $(\_)_i: \mathbb{RCS} \to \mathbb{G}$ from Proposition 5, then the predicate p is true componentwise if and only if it is true for $\mathcal{D}_i$ in $\mathbb{G}$ for all $i \in \{0, \dots, n\}$.

Another formulation is "*componentwise construction* of predicate p", where p is based on a certain universal property and thus transferring existence from universal constructions from $\mathbb{G}$ to $\mathbb{RCS}$. If e.g. p is the predicate for pushouts, componentwise construction of a $\mathbb{RCS}$-cospan $\mathbf{C} \xrightarrow{\mathbf{f}'} \mathbf{D} \xleftarrow{\mathbf{g}'} \mathbf{B}$ from a $\mathbb{RCS}$-span $\mathbf{C} \xleftarrow{\mathbf{g}} \mathbf{A} \xrightarrow{\mathbf{f}} \mathbf{B}$ consists of two steps: In a first step, one constructs pushout cospans $C_i \xrightarrow{f_i'} D_i \xleftarrow{g_i'} B_i$ of spans $C_i \xleftarrow{g_i} A_i \xrightarrow{f_i} B_i$ for each $i \in \{0, \dots, n\}$. In a second step one tries to define the projections d_j in $\mathbf{D} := (d_j : D_0 \rightharpoonup D_j)_{1 \le j \le n}$, cf. Definition 5, with the help of the pushouts' unique mediators. The cospan morphisms $\mathbf{f}'$ and $\mathbf{g}'$ consist of the respective components $(f_i')_{0 \le i \le n}$ and $(g_i')_{0 \le i \le n}$. The phrase "*p can be constructed componentwise*" then means that the newly constructed object $\mathbf{D}$ is an admissible object according to Definition 5, that the newly created morphisms $\mathbf{f}'$ and $\mathbf{g}'$ are admissible according to Definition 6, and that predicate p holds on the resulting diagram in $\mathbb{RCS}$, i.e. the square that arises from enhancing the above $\mathbb{RCS}$-span by the $\mathbb{RCS}$-cospan yields a pushout in $\mathbb{RCS}$. This procedure applies to other universal constructions in a similar way and after such a construction, we know that property p is valid componentwise.

"Commutativity" is valid componentwise by definition, but we also obtain

Proposition 6 *(Componentwise properties of $\mathbb{RCS}$). Morphism $\mathbf{f}: \mathbf{A} \to \mathbf{B}$ is a monomorphism if and only if it is such componentwise. Moreover, $\mathbb{RCS}$ has*

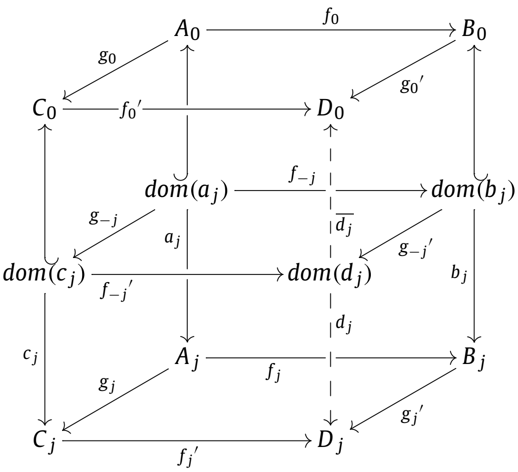
1. *all pullbacks,*
2. *all pushouts,*
3. *all coproducts,*

(is thus cocomplete), and they are constructed componentwise, respectively.

Proof. (1.) Componentwise validity of monomorphy and componentwise construction of pullbacks have been proven in [45] for so-called S-cartesian functor categories. We showed in [3] (see also [14]) that - for a certain schema category and using $\mathbb{G}$ as the underlying adhesive category - this functor category is equivalent to $\mathbb{RCS}$.

Thus, it remains to prove (2.) and (3.).

For the proof of (2.), let a span $(\mathbf{f}: \mathbf{A} \to \mathbf{B},\ \mathbf{g}: \mathbf{A} \to \mathbf{C})$ of $\mathbb{RCS}$-morphisms be given with $\mathbf{f} = (f_i : A_i \to B_i)_{0 \le i \le n}$ and $\mathbf{g} = (g_i : A_i \to C_i)_{0 \le i \le n}$. Resolving these two morphisms into a triple of $\mathbb{G}$-morphisms for each $j \in \{1, \dots, n\}$ as in (11) and constructing pushouts componentwise in $\mathbb{G}$, i.e. for f_0 and g_0, f_j and g_j, and for the span of resulting domain mappings f_{-j} and g_{-j}, see the dashed arrow in (11), yields two cubes on top of each other for each $j \in \{1, \dots, n\}$, shown in (13), in which the dashed vertical front right arrows d_j and $\overline{d_j}$ are unique mediators w.r.t. the middle pushout:



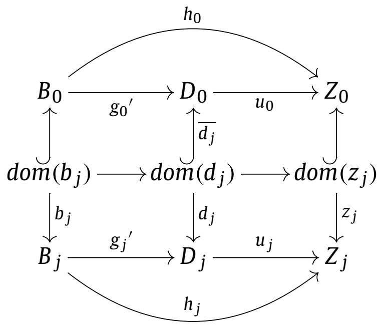
(13)

Because $\mathbb{G}$ is a hereditary pushout category by Proposition 2 and because the top face in the upper cube in (13) is a $\mathbb{G}$-pushout and the two back faces are pullbacks, cf. Definition 7, the prerequisite of the equivalent characterisation of hereditaryness in Proposition 3 are fulfilled. Hence the fact that the middle layer in (13) is also a pushout (by construction) implies that the two upper front faces become pullbacks and the vertical upward arrow $\overline{d_j}$ in the front right can be chosen to be an inclusion. This shows that the componentwise construction indeed yields an admissible comprehensive system

$$\mathbf{D} := (d_j : D_0 \rightharpoonup D_j)_{1 \le j \le n}$$

and a commutative square $\mathbf{g}' \circ \mathbf{f} = \mathbf{f}' \circ \mathbf{g}$ in $\mathbb{RCS}$. It remains to show that it is also a pushout.

Let for this a $\mathbb{RCS}$-object $\mathbf{Z} := (z_j : Z_0 \rightharpoonup Z_j)_{1 \le j \le n}$ and two $\mathbb{CS}$-morphisms $\mathbf{h} : \mathbf{B} \to \mathbf{Z}$ and $\mathbf{k} : \mathbf{C} \to \mathbf{Z}$ be given such that $\mathbf{h} \circ \mathbf{f} = \mathbf{k} \circ \mathbf{g}$. Then componentwise considerations easily yield unique $\mathbf{u} := (u_i : D_i \to Z_i)_{0 \le i \le n}$ factoring through the components of $\mathbf{h}$ and $\mathbf{k}$, see (14).[10]



(14)

It is easy to see that universality of d_j and $\overline{d_j}$ yield commutativity of all squares in (14) such that it remains to show that $\mathbf{u}$ is an $\mathbb{RCS}$-morphism. In particular, we have to show (8) as equivalence (commutativity (9) is already given). Let for this $x \in D_0$ be given. It is well known that pushouts in $\mathbb{G}$ yield jointly surjective cospans, i.e. x has a preimage y in C_0 or in B_0, cf. (13). Assume w.l.o.g. that there is $y \in B_0$ and ${g_0}'(y) = x$ (the case, where there is a preimage in C_0, is similar). Then again using (8) as equivalence (since $\mathbf{f}, \mathbf{g}, \mathbf{f}'$, and $\mathbf{g}'$ are reflective) several times yields

$$\begin{aligned} d_j(x) \text{ is defined } &\iff b_j(y) \text{ is defined} && (\text{because } \mathbf{g}' : \mathbf{B} \to \mathbf{D} \in \mathbb{RCS}^{\rightarrow}) \\ &\iff z_j(h_0(y)) \text{ is defined} && (\mathbf{h} : \mathbf{B} \to \mathbf{Z} \in \mathbb{RCS}^{\rightarrow}, \text{cf. } (14)) \\ &\iff z_j(u_0(x)) \text{ is defined} && (h_0 = u_0 \circ {g_0}' \text{ and } x = {g_0}'(y)), \end{aligned}$$

which shows that $\mathbf{u}$ is an $\mathbb{RCS}$-morphism.[11]

The proof of the existence of coproducts (3.) is similar: Let $(\mathbf{A}^x := (a^x_j : A^x_0 \rightharpoonup A^x_j)_{1 \le j \le n})_{x \in X}$ be a family of comprehensive systems indexed over some (possibly infinite) index set X. It is then easy to see that

$$\mathbf{A} := (\coprod_{x \in X} A^x_0 \xrightarrow{\coprod_{x \in X} a^x_j} \coprod_{x \in X} A^x_j)_{1 \le j \le n}$$

is the coproduct of them, where $\coprod_{x \in X} A^x_i$ denotes $\mathbb{G}$-coproducts (hence the $\mathbb{RCS}$-coproduct is constructed componentwise). For each j the partial morphism $\coprod_{x \in X} a^x_j$ is defined to be equal to a^y_j on each A^y_0 ($y \in X$). The unique mediator for a family ($\mathbf{f^x} : \mathbf{A^x} \to \mathbf{B}$) can be shown to be a $\mathbb{RCS}$-morphism by similar arguments as above for $\mathbf{u}$. It is well-known that all colimits can be constructed from binary pushouts and coproducts [46], hence $\mathbb{RCS}$ is indeed cocomplete. □

The equivalent characterisation of hereditaryness in Proposition 3 uses the predicates pushout, pullback, monomorphism, and commutativity, of which we have shown that validity in $\mathbb{RCS}$ is equivalent to componentwise validity. By jumping back and forth from a comprehensive system to its components, this yields

Corollary 1. $\mathbb{RCS}$ *is a hereditary pushout category.* □

Although it is not the focus of this paper, we mention another important consequence for the application of graph transformations in $\mathbb{RCS}$:

[10] The diagram in (14) depicts the situation for $\mathbf{h}$, for $\mathbf{k}$ the situation is the same.

[11] A more general proof has been given in [45], if $\mathbb{G}$ is a (variant of an) adhesive category, such that the result carries over to these base structures, as well.

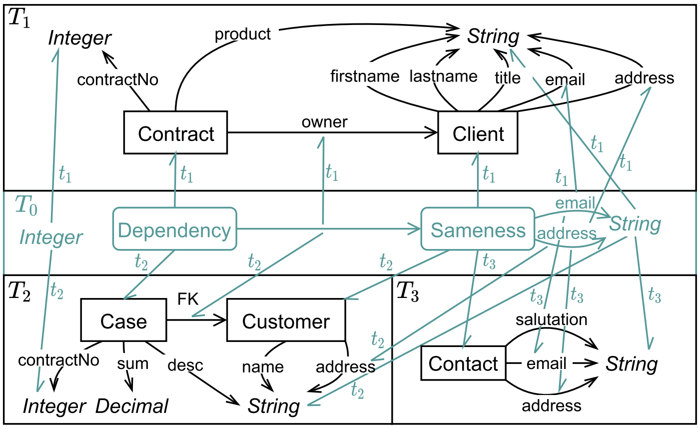


Fig. 7. Typing **T** for the running example.

Corollary 2. $\mathbb{RCS}$ *is a weak adhesive HLR category [27] w.r.t. the class of all monomorphisms.*

Proof. Heindel proves in [39, Prop. 8.1], that this conclusion can be drawn from Corollary 1, if pushouts are always stable under pullbacks, i.e. the implication "top face pushout, all side faces pullbacks $\Rightarrow$ bottom face pushout" holds for all choices of vertical morphisms in (3). But this implication is true in $\mathbb{RCS}$ by Proposition 6 and because this holds in $\mathbb{G}$ [18]. $\square$

This corollary guarantees validity of the classical theorems for graph rewriting [27] such as Local Church Rosser, Parallelism, or Local Confluence Theorem to hold in $\mathbb{RCS}$, as well.[12]

3.4. Typing

In Sect. 2.1, we mentioned that typing morphism are a common means to restrict a class of similarly structured models. In our running example, we only want to allow certain types of commonalities between different element types of systems components, e.g. "sameness" among `client/customer/contact` records and "dependency" of `cases` on `contracts` in Fig. 1. It is also important to specify *where* rules should be applied, e.g. that we want to update the `address` field, cf. Fig. 6. Thus, we need a typing mechanism.

The categorical interpretation of typed structures are *slice categories* $\mathbb{C} \downarrow T$ for an object $T \in |\mathbb{C}|$, called the *type* object. A slice category has morphisms $t^A : A \to T$, also called *instances*, with codomain T as objects. A morphism $f : t^A \to t^{A'}$ in this category is a morphism $f : A \to A' \in \mathbb{C}^{\to}$ between the respective domains that respects typing, i.e. $t^{A'} \circ f = t^A$ holds.

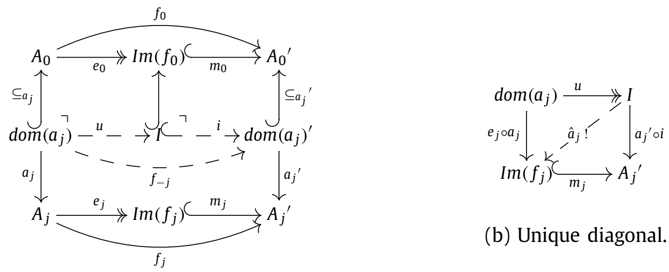
The comprehensive system **T** shown in Fig. 7 may serve as a suitable type object for the system **D** of our running example in Fig. 1. It combines the individual metamodels of $D_{1/2/3}$ together with the custom commonality types "sameness" and "dependency" together with commonalities witnessing common base type appearances (`String` and `Integer`). Moreover, there are three "edge"-commonalities to express relationships of `owner/FK` ($T_{1/2}$), `address` ($T_{1/2/3}$), and `email` ($T_{1/3}$) features. This demonstrates the usefulness or property (8), preservation of definedness, e.g. the fact that $t_3 : T_0 \rightharpoonup T_3$ is undefined on `Dependency` enforces **T**-instances to not have `Dependency`-typed commonalities, whose projections into the third component are defined: The concepts `case` and `contract` have no counterpart in the CRM. However, working with $\mathbb{RCS} \downarrow \mathbf{T}$ would impose too strong restrictions on the instances, i.e. the additional reflection property would enforce every commonality type to have a manifestation in every instance, which is practically unfeasible. Therefore, we have to combine $\mathbb{CS}$- and $\mathbb{RCS}$-morphism types into the following definition that only requires reflection of definedness for morphisms between instances and not for typing morphisms.

Definition 8 *(Typed comprehensive systems).* Let $\mathbf{T} \in |\mathbb{CS}|$ be a comprehensive systems. The category of typed comprehensive systems $\mathbb{TRCS}(\mathbf{T})$ over **T** has $\mathbf{t^C} : \mathbf{C} \to \mathbf{T} \in |\mathbb{CS} \downarrow \mathbf{T}|$ as objects and reflective morphisms $\mathbf{f} : \mathbf{C} \to \mathbf{C}' \in \mathbb{RCS}^{\to}$ as arrows such that $\mathbf{t^{C'}} \circ \mathbf{f} = \mathbf{t^C}$ commutes in $\mathbb{CS}$ (recall that $\mathbb{RCS} \subseteq \mathbb{CS}$).

We remark that $\mathbb{TRCS}(\mathbf{T}) = \mathcal{I} \downarrow \mathbf{T}$ is a general comma category, where $\mathcal{I} : \mathbb{RCS} \hookrightarrow \mathbb{CS}$ is the inclusion functor. One can show that all important properties are transferred to this category:

Proposition 7. *The category* $\mathbb{TRCS}(\mathbf{T})$ *has all pullbacks, pushouts and coproducts, which are stable under pullback.*

[12] Whereas we obtain this result as a corollary from hereditariness, it is proved directly for underlying adhesive categories in [45].



(a) Images componentwise.

(b) Unique diagonal.

Fig. 8. Images in $\mathbb{CS}$.

Proof. Is due to the fact that pushouts, pullbacks and coproducts are constructed componentwise in the same manner as in the proof for Proposition 6. It remains to show, that the componentwise constructions actually fall into $\mathbb{TRCS}(\mathbf{T})$, i.e. that compatible typing morphisms are retained. We are discussing the situation for pushouts:

Consider again the diagram in (13): Now, let $\mathbf{A}, \mathbf{B}, \mathbf{C}$ be domains of objects in $\mathbb{TRCS}(\mathbf{T})$ and $\mathbf{f}$ and $\mathbf{g}$ be $\mathbb{TRCS}(\mathbf{T})$-morphisms, i.e. there are additional $\mathbb{CS}$-morphisms $\mathbf{t^A} : \mathbf{A} \to \mathbf{T}$, $\mathbf{t^B} : \mathbf{B} \to \mathbf{T}$, $\mathbf{t^C} : \mathbf{C} \to \mathbf{T}$ such that $\mathbf{t^C} \circ \mathbf{g} = \mathbf{t^A}$ and $\mathbf{t^B} \circ \mathbf{f} = \mathbf{t^A}$. The fact that all horizontal faces in (13) are pushouts yields a family of morphisms $(t^D_\alpha : D_\alpha \to T_\alpha)_{\alpha \in \{-n,\ldots,-1,0,1,\ldots,n\}}$[13] such that $t^B_\alpha = t^D_\alpha \circ g'_\alpha$ and $t^C_\alpha = t^D_\alpha \circ f'_\alpha$. It remains to show that $t^D_0 \circ \overline{d_j} = \subseteq_{t_j} \circ t^D{}_{-j}$ and $t^D_j \circ d_j = t_j \circ t^D{}_{-j}$, which immediately follows by using the jointly epimorphism property of the middle pushout, combined with morphism property of $\mathbf{f}'$ and $\mathbf{g}'$ (commutativity (9)), and the commutative triangles, mentioned above.

The proof for coproducts works analogously and for pullbacks it is almost trivial since typing is retained by simple postcomposition. □

4. Result: pushouts along partial maps of comprehensive systems

The goal of this section is to prove that SPO rewriting is well possible for (typed) comprehensive systems by showing that the categories $\mathtt{Par}(\mathbb{RCS})$ and $\mathtt{Par}(\mathbb{TRCS}(\mathbf{T}))$ possess all pushouts. This will follow mainly from a result of Hayman and Heindel:

Proposition 8 *(Existence of pushouts of partial maps, [37]). Let $\mathbb{C}$ be a category with pullbacks in which for each span $C \stackrel{g}{\leftarrow} A \stackrel{f}{\rightarrow} B$ of morphisms there is a cospan $C \stackrel{f'}{\rightarrow} D \stackrel{g'}{\leftarrow} B$ making the resulting square commutative. $\mathtt{Par}(\mathbb{C})$ has all pushouts if and only if $\mathbb{C}$ is a hereditary pushout category and inverse image functions have upper adjoints.* □

Proposition 9. *$\mathbb{RCS}$ and $\mathbb{TRCS}(\mathbf{T})$ have images and the pullback functors preserve them.*

Proof. Let $\mathbf{f} : \mathbf{A} \to \mathbf{A}' = (f_i : A_i \to A_i{}')_{0 \le i \le n}$ be a $\mathbb{CS}$-arrow. We use $\mathbb{G}$'s epi-mono-factorisations [18] to decompose f_0 and $(f_j)_{1 \le j \le n}$ accordingly. In particular $f_0 = m_0 \circ e_0$. Then the pullback of m_0 and $\subseteq_{a_j}{}'$ and its unique mediator u w.r.t. f_{-j} and $e_0 \circ \subseteq_{a_j}$ yields the situation in Fig. 8a, where the left upper square is a pullback by the pullback decomposition lemma.

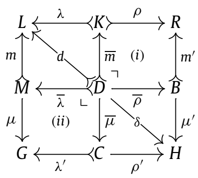
In $\mathbb{G}$ pullbacks preserve epimorphisms, i.e. u is an epimorphism and the square in Fig. 8b has a unique diagonal [18] $\hat{a}_j : I \to Im(f_j)$, such that everything commutes. Adding this diagonal in Fig. 8a yields $\mathbf{Im(f)} := (Im(f_0) \stackrel{\hat{a}_j}{\longleftarrow} Im(f_j))_{1 \le j \le n}$ and the inclusion $\mathbf{m} : \mathbf{Im(f)} \hookrightarrow \mathbf{A}'$. Moreover, $\mathbf{Im(f)}$ can be shown to be the image of $\mathbf{f} : \mathbf{A} \to \mathbf{A}'$ in $\mathbb{RCS}$, because it was set up by componentwise epi-mono-factorisation (in $\mathbb{G}$), in which the mono-part is componentwise the least subobject of the respective codomains of f_0 and f_j. Images in $\mathbb{TRCS}(\mathbf{T})$ are constructed in the same way, i.e. if $\mathbf{t^A} : \mathbf{A} \to \mathbf{T}$ and $\mathbf{t^{A'}} : \mathbf{A}' \to \mathbf{T}$ are $\mathbb{TRCS}(\mathbf{T})$ objects, we get a typing morphism $\mathbf{t^{Im(f)}} : \mathbf{Im(f)} \to \mathbf{T}$ by simply composing $\mathbf{t^{A'}} \circ \mathbf{m}$.

Pullback functors preserve images in $\mathbb{RCS}$ and $\mathbb{TRCS}(\mathbf{T})$ because of the essential uniqueness of epi-mono-factorisations, preservation of monomorphisms and epimorphisms [18] under pullbacks in $\mathbb{G}$, and componentwise pullback construction (cf. Proposition 6). □

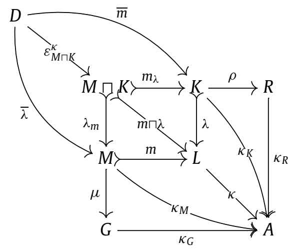
Theorem 1. *$\mathtt{Par}(\mathbb{RCS})$ and $\mathtt{Par}(\mathbb{TRCS}(\mathbf{T}))$ have all pushouts.*

Proof. Firstly, $\mathbb{RCS}$ has all pushouts by Proposition 6 ($\mathbb{TRCS}(\mathbf{T})$ due to Proposition 7 resp.) and thus span-completions. Secondly, $\mathbb{RCS}$ and $\mathbb{TRCS}(\mathbf{T})$ have images (Proposition 9) and coproducts, therefore inverse image functions have upper adjoints (Proposition 4). Hence, we can apply Proposition 8 to yield the desired property. □

[13] Let $D_{-j} := dom(d_j)$ and $T_{-j} := dom(t_j)$.



(a) Gluing diagram.



(b) Domain of definition of the diagonal.

Fig. 9. Pushout construction in $\mathtt{Par}(\mathbb{C})$.

5. Conflict-freeness and relationship with DPO

We begin with an analysis of $\mathtt{Par}(\mathbb{C})$-pushouts in terms of an elementary construction in the underlying category $\mathbb{C}$. This construction has originally been formulated for graph-based structures using set-theoretic concepts in [25] and was later stated purely categorically in [37]. Michael Löwe also developed an even more general (pushout-like) *gluing construction* of morphisms in arbitrary span categories [35,36]. Fig. 9a depicts a gluing construction diagram for the $\mathtt{Par}(\mathbb{C})$-pushout from Fig. 3b in Definition 1:

The squares (i) and (ii) are pullbacks, the bottom right square is a pushout, and δ is its diagonal. Special attention goes to D, the domain of definition of the pushout diagonal $[d, \delta\rangle : L \rightharpoonup H$ in $\mathtt{Par}(\mathbb{C})$. Intuitively, it is the "biggest" subobject of L, on which the domain restrictions of both ρ and μ are totally defined and that is also a subobject of the intersection of the domains of definition of $[m, \mu\rangle$ and the rule $[\lambda, \rho\rangle$ [25]. Categorically, cf. [37], D is given by applying the comonad $\kappa^{-1}\forall_\kappa : Sub(L) \to Sub(L)$ w.r.t. the adjunction $\kappa^{-1} \dashv \forall_\kappa$ (Definition 4) on $[m] \sqcap [\lambda] \in Sub(L)$, which is given by the pullback (intersection) of m and λ. The morphism $\kappa := \kappa_L$ is constructed by calculating the colimit $(A, (\kappa_X : X \to A)_{X\in\{G,M,L,K,R\}})$ of the diagram (μ, m, λ, ρ) in $\mathbb{C}$ by subsequent pushout applications, see Fig. 9b. In [37], it was shown that, for this constructed D, one also obtains $\forall_\rho[\overline{m}] = [m']$ and $\forall_\mu[\overline{\lambda}] = [\lambda']$.

In Definition 1, we introduced an SPO-derivation as a single pushout in a category of partial morphisms $\mathtt{Par}(\mathbb{C})$ along a *total match* morphism μ. Thus, m is the identity on L, and therefore $M = L$ and $[m] \sqcap [\lambda] = [\lambda]$. The match μ is said to be *conflict-free* when the co-match $[m', \mu'\rangle$ is total, i.e. m' is an isomorphism. It is desirable to have a concrete characterisation of conflict-freeness, i.e. for m' being an isomorphism in $\mathbb{RCS}$.

Indeed, there exists a characterisation in $\mathbb{G}$: A match $\mu : L \to G$ w.r.t. a rule $[\lambda, \rho\rangle : L \rightharpoonup R$ is *conflict-free*, if and only if the following statement holds:

$$\forall x, y \in L : \mu(x) = \mu(y) \implies (x \in K \Leftrightarrow y \in K), \tag{15}$$

see [25]. Our goal is to show that (15) characterises conflict-freeness also in $\mathbb{RCS}$, although the involved construction of upper adjoints can, in general, not be carried out componentwise. Recall that we can choose involved monomorphisms $\boldsymbol{\lambda}, \mathbf{m}', \overline{\mathbf{m}}$, etc as inclusions. Again, we use capital letters to denote the respective domains of partial maps, cf. Sect. 2.4.

Theorem 2 *(Conflict-freeness). Let in $\mathbb{RCS}$ a linear rule $[\boldsymbol{\lambda}, \boldsymbol{\rho}\rangle : \mathbf{L} \rightharpoonup \mathbf{R}$ be given. A total match $\boldsymbol{\mu} : \mathbf{L} \to \mathbf{G}$ is conflict-free, if and only if (15) holds for $\boldsymbol{\mu}$ in $\mathbb{RCS}$.*[14]

Proof. "⇒": Assume that $\boldsymbol{\mu}$ is conflict-free, i.e. $\boldsymbol{\mu}'$ is total. Thus, $\mathbf{m}'$ can be chosen as identity and it follows that $\mathbf{B} = \mathbf{R}$ and $\mathbf{D} = \mathbf{K}$. Now assuming that the implication in (15) does not hold, i.e. $\exists x, y \in \mathbf{L} : \boldsymbol{\mu}(x) = \boldsymbol{\mu}(y)$ and w.l.o.g. $x \in \mathbf{K} \wedge y \notin \mathbf{K}$. Now, when constructing the colimit $\mathbf{A}$ of the spans $(\boldsymbol{\lambda}, \boldsymbol{\rho})$ and $(\mathbf{m}, \boldsymbol{\mu})$ the morphism $\boldsymbol{\kappa} : \mathbf{L} \to \mathbf{A}$ maps x and y to the same element because $\boldsymbol{\mu}$ does. Let $z := \boldsymbol{\kappa}(x) = \boldsymbol{\kappa}(y)$. Thus $z \notin \forall_{\boldsymbol{\kappa}}\lambda$ due to the definition of $\forall_{\boldsymbol{\kappa}}$, cf. Proposition 4, and therefore $x \notin \mathbf{D}$. But this is a contradiction since we already had $\mathbf{D} = \mathbf{K}$ and $x \in \mathbf{K}$.

"⇐": The construction in Fig. 9b shows that validity of the implication (15) carries over to $\boldsymbol{\kappa}$, because $\boldsymbol{\rho}$ is a monomorphism, thus

$$\forall x, y \in \mathbf{L} : \boldsymbol{\kappa}(x) = \boldsymbol{\kappa}(y) \Rightarrow (x \in \mathbf{K} \iff y \in \mathbf{K}). \tag{16}$$

In the sequel, we refer to objects and morphisms in Figs. 9a, 9b. To show that $\mathbf{m}'$ is an isomorphism, we have to take a closer look at the construction of the upper adjoint $\forall_{\boldsymbol{\kappa}}\boldsymbol{\lambda}$, when (16) is valid. There is no larger subobject of $\mathbf{A}$, which pulls back to a subobject of $\boldsymbol{\lambda}$, than the one, which arises from componentwise construction of upper adjoints in $\mathbb{G}$, i.e. by calculating $\forall_{\kappa_i}\lambda_i$ for all $-n \le i \le n$, where κ_{-j}/λ_{-j} are the restrictions of κ_0/λ_0 to the domains of definition of l_j/k_j. By (6) (and surrounding remarks) and (16) there are resulting pullback squares for all $1 \le j \le n$, see Fig. 10.

[14] A statement $x \in \mathbf{K}$ for some comprehensive system $\mathbf{K}$ means: x is an element of a component K_j or a commonality representative in K_0.

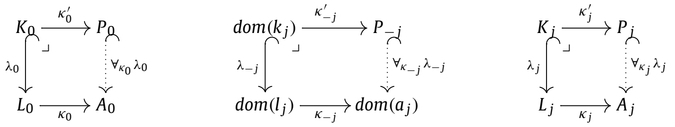


Fig. 10. Componentwise construction of upper adjoints possibly yields no $\mathbb{RCS}$-morphism $\forall_{\boldsymbol{\kappa}}\boldsymbol{\lambda}$ (dotted lines).

Because $\boldsymbol{\kappa}$ is a reflective morphism, the universal property of $\forall_{\kappa_0}$, being the right-adjoint of the pullback functor in $\mathbb{G}$ (see Sect. 2.4), yields unique partial $\mathbb{G}$-morphisms p_j in the comprehensive system $\mathbf{P} = (p_j : P_0 \rightharpoonup P_j)$. E.g. the arrow $\subseteq$ in

$$\begin{array}{ccc} K_0 & \xrightarrow{\kappa'_0} & P_0 \\ \uparrow & & \uparrow \subseteq \\ dom(k_j) & \xrightarrow[\kappa'_{-j}]{} & P_{-j} \end{array} \tag{17}$$

arises due to the universal property of the right-adjoint.

Universality also yields $\mathbb{CS}$-morphism $\forall_{\boldsymbol{\kappa}}\boldsymbol{\lambda} : \mathbf{P} \to \mathbf{A}$. However, this arrow must not necessarily reflect definedness, see the dotted lines in Fig. 10.

To achieve reflection of definedness, one reduces P_0 to the largest subalgebra P'_0, such that the restriction of this arrow becomes reflective. To define P'_0, one has to remove elements z from P_0, for which there is j, such that p_j is undefined, but a_j is defined, together with all z' which are mapped to z by a sequence *op* of applications of functions in P_0, i.e.: If z is thus removed, every z', for which $z = op(z')$ must also be removed. Clearly, definedness of a partial morphism on some element z' implies definedness also on $z = op(z')$, i.e. we remove only elements, where p_j is undefined. Thus, the graphs $(P_j)_{j\geq 1 \text{ or } j \leq -1}$ remain unchanged.

Assume now that $\mathbf{m}'$ in Fig. 9a is not an isomorphism, then, by the definition of upper adjoint (Definition 4), $\overline{\mathbf{m}}$ is also not surjective. Hence there is $y \in \mathbf{K} \setminus \mathbf{D}$. Because P_j are not reduced for $j \geq 1$, we have $K_j = D_j$, see Fig. 10, hence $y \in K_0 \setminus D_0$. Let

$$z := \kappa_0(y),$$

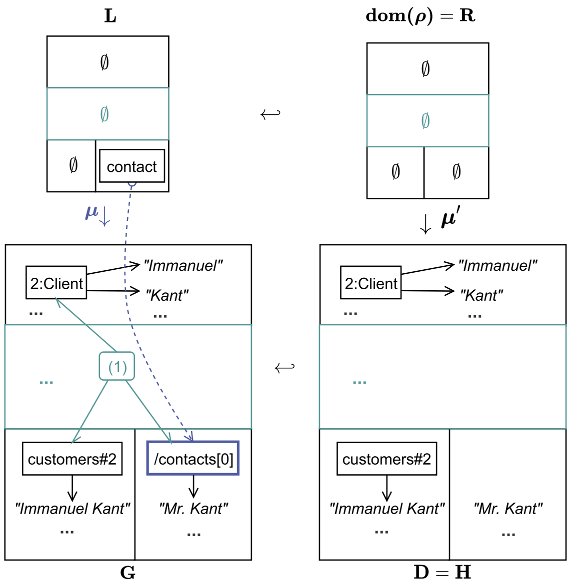
then $z \in P_0$ (left pullback square in Fig. 10). Assume $z \in P'_0$, i.e. $p_j(z)$ is defined if and only if $a_j(z)$ is defined. Since $d_0 : D_0 \to L_0$ arises as the component with index 0 of the pullback of $\forall_{\boldsymbol{\kappa}}\boldsymbol{\lambda}$ along κ_0, we must have $y \in D_0$, a contradiction.

Thus, $z \in P_0 \setminus P'_0$ such that z must have been removed from P_0 when creating P'_0. But then, by the above remarks, $p_j(z)$ must be undefined, i.e. $z \notin P_{-j}$. Hence, (17) yields $y \notin dom(k_j)$ such that $k_j(y)$ is undefined and thus: (I) $a_j(z)$ is undefined, because $\boldsymbol{\kappa} \circ \boldsymbol{\lambda} : \mathbf{K} \to \mathbf{A}$ is reflective. Then, z can only have been removed due to some z', for which $z = op(z')$ and $p_j(z')$ undefined and (II) $a_j(z')$ is defined, see the construction above. But this contradicts the fact that a_j is a partial $\mathbb{G}$-morphism, because then $z = op(z')$ yields (II) $\Rightarrow \neg$ (I). $\square$

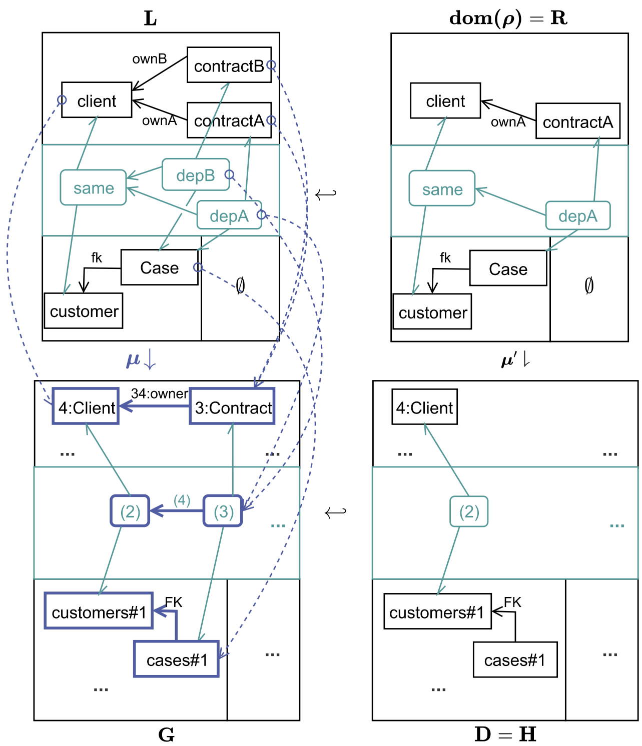
SPO at conflict-free matches generalises DPO and SqPO with linear left-hand sides: When a match is conflict-free, the morphisms $m, \overline{m}, m'$ in Fig. 9a are isomorphisms and likewise can be chosen as identities since all constructions are only up to isomorphism. Hence, the upper row in Fig. 9a collapses. The square (ii) is a pullback, more concretely the chain of morphisms $(\lambda', \overline{\mu})$ is constructed as a mono-final pullback complement [39], which coincides with the pushout complement [34] if the latter exists. The square with diagonal δ is a pushout by construction and therefore the gluing diagram in Fig. 9a becomes the diagram of a SqPO or DPO derivation, cf. Fig. 3a.

Finally, we demonstrate the differences between SPO, SqPO and DPO at a concrete example. Fig. 11 shows two rule applications that are possible in SPO but not in DPO because in both cases there exists no pushout complement (because the *dangling* (Fig. 11a) and *identification* (Fig. 11b) conditions [27] are violated). The rule in Fig. 11a demonstrates how the application of a rule, which specifies the deletion of a `contact`-object in the CRM (e.g. cause a customer opts out from receiving more marketing material), leads to a deletion of incident edges, including a deletion of the commonality (1), otherwise the inclusion $\mathbf{H} \hookrightarrow \mathbf{G}$ would not be reflective. The match $\boldsymbol{\mu}$ in Fig. 11a is conflict-free and the rule could also be interpreted as a SqPO-rule leading to the same result. The rule application in Fig. 11b shows a match $\boldsymbol{\mu}$ that is not conflict-free, i.e. $\boldsymbol{\mu}'$ is partial. The rule specifies a deletion of `contracts` when there is a `case` depending on two `contracts` (e.g. due to removing ambiguity). Constructing the $\mathtt{Par}(\mathbb{RCS})$-pushout for this rule and the non-injective $\boldsymbol{\mu}$ results in deleting both `contracts` together with associated commonalities, an effect known as *"precedence of deletion over preservation"*. As the latter effect may be undesired in practice, matches are sometimes required to be monomorphisms as well [47]. Moreover, this SPO rule application can not be interpreted as an SqPO rule application, because SqPO requires final pullback complements when applying the left leg of a rule, but the four morphisms in Fig. 11b obviously do not form a pullback.

As a conclusion, SPO is much more *"liberal"* compared to the more restrictive DPO when it comes to rule application, which, however, sometimes may lead to undesired results.



(a) Deletion in unknown context



(b) Precedence of deletion over preservation

Fig. 11. Non applicable DPO rules.

6. Conclusion

We introduced the category $\mathbb{CS}$ of *Comprehensive Systems* which is basically a functor category invented in [8] and generalised in [45], its basic ideas originating from the theory of triple graphs [4]. A $\mathbb{CS}$-object represents an all-embracing view on a software system of possibly heterogeneously typed components, in which all inter-model relationships are internalised and where "partiality" is the crucial methodology allowing to collect (possibly different kinds of) commonalities

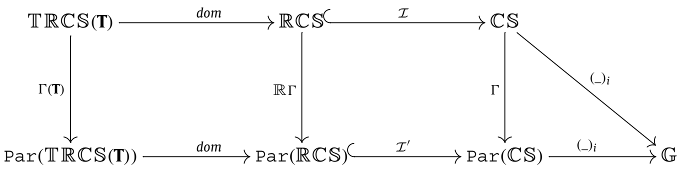


Fig. 12. Overview over categories.

between components into a single component. As long as the parameter n is chosen large enough, the overall schema of $\mathbb{CS}$-objects remains constant independent of the system landscape under consideration.

Despite the presence of heterogeneous typings, a universal treatment of these systems can be achieved by representing the components as graph-like structures, i.e. objects of a presheaf topos category $\mathbb{G}$, a category based on signatures with unary operation symbols only. In contrast to "fully" partial algebras, comprehensive systems allow partiality only for the inter-relation between commonalities and components, whereas the algebraic inner component structure remains total.

Graph transformations and especially the SPO approach rely on the existence of (certain) pushouts, which do not exist in $\mathtt{Par}(\mathbb{CS})$ even in simple examples. Hence, we narrowed the universe of discourse from $\mathbb{CS}$ to the subcategory $\mathbb{RCS}$, which has the same objects as $\mathbb{CS}$, but morphisms are claimed to reflect definedness of the partial maps within the systems. Typed systems $\mathbb{TRCS}(\mathbf{T})$ over a fixed type system $\mathbf{T}$ were introduced alongside.

In order to investigate the applicability of the SPO technique for comprehensive systems, the categories $\mathtt{Par}(\mathbb{C})$, whose morphisms serve as SPO rules, were introduced. Comprehensive systems were put into this context. The complete picture of all mentioned categories is depicted in Fig. 12, where *dom* is the usual domain functor out of comma categories, $\mathcal{I}, \mathcal{I}'$ are inclusion functors, $\mathbb{R}\Gamma$ is the restriction of the graphing functor to $\mathbb{RCS}$, and $\Gamma(\mathbf{T})$ analogously embeds typed systems.

We proved the following main results:

- $\mathbb{RCS}$ is a hereditary pushout category and hence also a weak adhesive HLR category, i.e. fundamental results about parallelism and confluence are valid.
- $\mathtt{Par}(\mathbb{RCS})$ and $\mathtt{Par}(\mathbb{TRCS}(\mathbf{T}))$ possess all pushouts and qualify for successful application of SPO rewriting in $\mathbb{RCS}$ and $\mathbb{TRCS}(\mathbf{T})$.
- The set-theoretic characterisation of conflict-freeness carries over to SPO rewriting in $\mathbb{RCS}$.
- Definedness-reflecting (closed) morphisms inherently contain negative application condition facets.

7. Related and future work

The best reference for *Single Pushout Rewriting* is [1], see also [48]. Pushouts in partial map categories and especially hereditariness of colimits have been thoroughly investigated in [37,40].

Our approach still lacks the proof that it is practically useful, but we hope that SPO rules can serve as a basis for repair rules [45,49] in order to maintain consistency of multimodels [43,7]. Another important aspect is a detailed analysis of *attributes* and computations on them, which is often necessary in practice. Though, we conjecture that the results of this paper still hold when we exchange $\mathbb{G}$ with attributed graphs, it may be worthwhile to look at possibilities that avoid "copying" the algebra part n-times for every component of a comprehensive system, possibly by borrowing ideas from [50].

The strength of "intrinsic" partiality in algebraic structures is nothing new [51]. These old insights not only helped us to prove SPO applicability in comprehensive systems, but, in former papers, we were also able to generalise the dynamical behaviour of triple graph grammars and graph diagram grammars [3,14], which was already mentioned in the introduction. Michael Löwe recognised these strengths especially in [25], e.g. the simplicity of *initial* graph-like structures, but he was also interested in partial algebras in later years: [2] demonstrates that he continued to work in that area. There is, however, a subtle, but important difference between partial algebras and comprehensive systems: Whereas the former allow partiality for all operations, the latter allow partiality only for the commonality definitions but not for the inner component structures, such that general results for partial algebras not always carry over.

Compared to the previous conference version, we now provided an analysis of conflict-freeness and accounted for typed comprehensive systems. Still, more properties of graph rewriting of comprehensive systems remain to be investigated. E.g. we have to consider comprehensive systems as an indexed category $\mathbb{N} \to \mathbb{CAT}$, i.e. we want to investigate the behaviour, which arises when n is varied. The situation is as in the following quotation: "The contents of this [paper] should rather be considered a starting point ... than the final document of this research issue".[15]

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[15] This is an almost identical citation of the last statement in Michael's PhD Thesis!

Acknowledgements

We want to thank Tobias Heindel for pointing to the basic theoretical results needed to produce the theorems in this paper. Especially, we found his categorical analysis and generalisation of results concerning "pushouts of partial maps" very useful. Thanks also go to the anonymous referees for many valuable comments and suggestions.

References

[1] M. Löwe, Extended algebraic graph transformation, Ph.D. thesis, Technical University of Berlin, Germany, 1991, http://d-nb.info/910935696.

[2] M. Löwe, M. Tempelmeier, Single-pushout rewriting of partial algebras, in: D. Plump (Ed.), Proceedings of GCM Co-Located with ICGT/STAF, L'Aquila, Italy, in: CEUR Workshop Proceedings, vol. 1403, 2015, pp. 82–96, http://ceur-ws.org/Vol-1403/paper7.pdf.

[3] P. Stünkel, H. König, Y. Lamo, A. Rutle, Towards multiple model synchronization with comprehensive systems, in: H. Wehrheim, J. Cabot (Eds.), Fundamental Approaches to Software Engineering, in: Lecture Notes in Computer Science, vol. 12076, Springer, Cham, 2020, pp. 335–356.

[4] A. Schürr, Specification of graph translators with triple graph grammars, in: E.W. Mayr, G. Schmidt, G. Tinhofer (Eds.), Graph-Theoretic Concepts in Computer Science, Springer, Berlin, Heidelberg, 1995, pp. 151–163.

[5] F. Trollmann, S. Albayrak, Extending model to model transformation results from triple graph grammars to multiple models, in: D. Kolovos, M. Wimmer (Eds.), Theory and Practice of Model Transformations, Springer International Publishing, Cham, 2015, pp. 214–229.

[6] F. Trollmann, S. Albayrak, Extending model synchronization results from triple graph grammars to multiple models, in: P. Van Gorp, G. Engels (Eds.), Theory and Practice of Model Transformations, Springer International Publishing, Cham, 2016, pp. 91–106.

[7] S. Feldmann, K. Kernschmidt, M. Wimmer, B. Vogel-Heuser, Managing inter-model inconsistencies in model-based systems engineering: application in automated production systems engineering, J. Syst. Softw. 153 (2019) 105–134, https://doi.org/10.1016/j.jss.2019.03.060.

[8] P. Stünkel, H. König, Y. Lamo, A. Rutle, Multimodel correspondence through inter-model constraints, in: Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Programming'18 Companion, Association for Computing Machinery, New York, NY, USA, 2018, pp. 9–17.

[9] G. Spanoudakis, A. Zisman, Inconsistency management in software engineering: survey and open research issues, in: Handbook of Software Engineering and Knowledge Engineering, 2001, pp. 329–380.

[10] C. Nentwich, W. Emmerich, A. Finkelsteiin, Consistency management with repair actions, in: ICSE '03, 2003, pp. 455–464.

[11] N. Drivalos, D.S. Kolovos, R.F. Paige, K.J. Fernandes, Engineering a DSL for software traceability, in: D. Gašević, R. Lämmel, E. Van Wyk (Eds.), Software Language Engineering, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2009, pp. 151–167.

[12] L. Samimi-Dehkordi, B. Zamani, S. Kolahdouz-Rahimi, EVL+Strace: a novel bidirectional model transformation approach, Inf. Softw. Technol. 100 (2018) 47–72, https://doi.org/10.1016/j.infsof.2018.03.011.

[13] P. Stevens, Bidirectional transformations in the large, in: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, MODELS, 2017, pp. 1–11.

[14] P. Stünkel, H. König, Y. Lamo, A. Rutle, Towards multiple model synchronization with comprehensive systems: extended version, Tech. rep., University of Applied Sciences, FHDW Hannover, 2020, https://fhdwdev.ha.bib.de/public/papers/02020-01.pdf.

[15] H. König, P. Stünkel, Single pushout rewriting in comprehensive systems, in: F. Gadducci, T. Kehrer (Eds.), Graph Transformation, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2020, pp. 91–108.

[16] F. Gadducci, T. Kehrer (Eds.), Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Bergen, Norway, June 25-26, 2020, Proceedings, Lecture Notes in Computer Science, vol. 12150, Springer, 2020.

[17] E. Biermann, C. Ermel, G. Taentzer, Precise semantics of EMF model transformations by graph transformation, in: K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, M. Völter (Eds.), MODEL 2008, Springer, Berlin, Heidelberg, 2008, pp. 53–67.

[18] R. Goldblatt, Topoi: The Categorial Analysis of Logic, revised edition, Dover Publications, 2006.

[19] D. Plump, Hypergraph rewriting: critical pairs and undecidability of confluence, in: R. Sleep, R. Plasmeijer, M. van Eekelen (Eds.), Term Graph Rewriting, John Wiley, 1993, pp. 201–213.

[20] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, F. Zanasi, Confluence of graph rewriting with interfaces, in: H. Yang (Ed.), Programming Languages and Systems, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2017, pp. 141–169.

[21] H. Ehrig, M. Löwe, Categorical principles, techniques and results for high-level-replacement systems in computer science, Appl. Categ. Struct. 1 (1) (1993) 21–50, https://doi.org/10.1007/BF00872984.

[22] H. Ehrig, U. Prange, G. Taentzer, Fundamental theory for typed attributed graph transformation, in: H. Ehrig, G. Engels, F. Parisi-Presicce, G. Rozenberg (Eds.), Graph Transformations, Springer, Berlin, Heidelberg, 2004, pp. 161–177.

[23] Object Management Group, Unified Modeling Language (UML) v.2.4.1, http://www.omg.org/spec/UML, 2015.

[24] Object Management Group, Meta Object Facility (MOF) core specification v. 2.4.1, http://www.omg.org/spec/MOF, 2016.

[25] M. Löwe, Algebraic approach to single-pushout graph transformation, Theor. Comput. Sci. 109 (1) (1993) 181–224, https://doi.org/10.1016/0304-3975(93)90068-5.

[26] H. Ehrig, U. Prange, Weak adhesive high-level replacement categories and systems: a unifying framework for graph and Petri net transformations, in: K. Futatsugi, J.-P. Jouannaud, J. Meseguer (Eds.), Algebra, Meaning, and Computation: Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 235–251.

[27] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, 1st edition, Springer-Verlag, Berlin, Heidelberg, 2006.

[28] G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, World Scientific, 1997.

[29] H. Ehrig, M. Pfender, H.J. Schneider, Graph-grammars: an algebraic approach, in: 14th Annual Symposium on Switching and Automata Theory, SWAT 1973, 1973, pp. 167–180.

[30] H. Ehrig, A. Habel, H.-J. Kreowski, F. Parisi-Presicce, From graph grammars to high level replacement systems, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1991, pp. 269–291.

[31] S. Lack, P. Sobociński, Adhesive categories, in: I. Walukiewicz (Ed.), Foundations of Software Science and Computation Structures, Springer, Berlin, Heidelberg, 2004, pp. 273–288.

[32] J.C. Raoult, On graph rewritings, Theor. Comput. Sci. 32 (1) (1984) 1–24, https://doi.org/10.1016/0304-3975(84)90021-5.

[33] R. Kennaway, Graph rewriting in some categories of partial morphisms, in: H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Grammars and Their Application to Computer Science, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 1991, pp. 490–504.

[34] A. Corradini, T. Heindel, F. Hermann, B. König, Sesqui-pushout rewriting, in: A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.), Graph Transformations, Springer, Berlin, Heidelberg, 2006, pp. 30–45.

[35] M. Löwe, Graph rewriting in span-categories, in: H. Ehrig, A. Rensink, G. Rozenberg, A. Schürr (Eds.), Graph Transformations, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 218–233.

[36] M. Löwe, Refined graph rewriting in span-categories, in: H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Graph Transformations, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 111–125.
[37] J. Hayman, T. Heindel, On pushouts of partial maps, in: H. Giese, B. König (Eds.), Graph Transformation, Springer International Publishing, Cham, 2014, pp. 177–191.
[38] E. Robinson, G. Rosolini, Categories of partial maps, Inf. Comput. 79 (2) (1988) 95–130, https://doi.org/10.1016/0890-5401(88)90034-X.
[39] T. Heindel, A category theoretical approach to the concurrent semantics of rewriting: adhesive categories and related concepts, Ph.D. thesis, University of Duisburg-Essen, Apr 2010, https://duepublico2.uni-due.de/receive/duepublico_mods_00022389.
[40] T. Heindel, Hereditary pushouts reconsidered, in: H. Ehrig, A. Rensink, G. Rozenberg, A. Schürr (Eds.), Graph Transformations, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 250–265.
[41] H. Ehrig, U. Golas, F. Hermann, Categorical frameworks for graph transformation and HLR systems based on the DPO approach, Bull. Eur. Assoc. Theor. Comput. Sci. 3 (102) (Sep. 2013) 102.
[42] H. König, Trouble with wrong adjoints, Techreport 5, FHDW Hannover, Freundallee 15, 30173 Hannover, Germany, 2012.
[43] Z. Diskin, H. König, M. Lawford, Multiple model synchronization with multiary delta lenses with amendment and K-Putput, Form. Asp. Comput. 31 (5) (2019) 611–640, https://doi.org/10.1007/s00165-019-00493-0.
[44] R. Heckel, A. Wagner, Ensuring consistency of conditional graph grammars - a constructive approach, Electron. Notes Theor. Comput. Sci. 2 (1995) 118–126, https://doi.org/10.1016/S1571-0661(05)80188-4.
[45] J. Kosiol, L. Fritsche, A. Schürr, G. Taentzer, Adhesive subcategories of functor categories with instantiation to partial triple graphs, in: E. Guerra, F. Orejas (Eds.), Graph Transformation, in: Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 38–54.
[46] J. Adámek, H. Herrlich, G. Strecker, Abstract and Concrete Categories: The Joy of Cats, Pure and Applied Mathematics, Wiley, 1990.
[47] A. Habel, K.-H. Pennemann, Correctness of high-level transformation systems relative to nested conditions, Math. Struct. Comput. Sci. 19 (2) (2009) 245–296, https://doi.org/10.1017/S0960129508007202.
[48] P. Burmeister, M. Monserrat, F. Rosselló, G. Valiente, Algebraic transformation of unary partial algebras ii: single-pushout approach, Theor. Comput. Sci. 216 (1) (1999) 311–362, https://doi.org/10.1016/S0304-3975(97)00282-X.
[49] F. Rabbi, Y. Lamo, I.C. Yu, L.M. Kristensen, A diagrammatic approach to model completion, in: J. Dingel, S. Kokaly, L. Lucio, R. Salay, H. Vangheluwe (Eds.), Proceedings of the 4th Workshop on the Analysis of Model Transformations Co-Located with the 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015, Ottawa, Canada, September 28, 2015, in: CEUR Workshop Proceedings, CEUR-WS.org, vol. 1500, 2015, pp. 56–65.
[50] P. Schultz, D.I. Spivak, C. Vasilakopoulou, R. Wisnesky, Algebraic databases, arXiv:1602.03501, 2016.
[51] P. Burmeister, Partial algebras — an introductory survey, in: Algebras and Orders, Springer, Netherlands, 1993, pp. 1–70.