SPO-Rewriting of Constrained Partial Algebras

Michael Löwe

FHDW Hannover, Freundallee 15, 30173 Hannover

Abstract Recently, single-pushout rewriting (SPO) has been applied to arbitrary partial algebras (PA). On the one hand, this allows a simple and straightforward integration of (base type) attributes into graph transformation. On the other hand, SPO-PA-rewriting comes equipped with an easy-to-check application condition, namely that an operation cannot be defined twice on the same set of arguments. This provides very natural termination criteria for example in model transformation.
In this paper, we generalise this approach to constrained partial algebras. We allow two different types of constraints, namely (i) requiring some operations to be total and (ii) enforcing some consistency conditions on the algebras by suitable conditional equations. We show that this generalisation again induces an easy to check application condition and provides considerably more expressive power: For example, constraints allow a straightforward algebraic model for the object-oriented concept of inheritance with runtime specialisation and generalisation of objects.

1 Introduction

In [18,20], we introduced single-pushout rewriting (SPO) in categories of partial morphisms over partial algebras (PA) wrt. *arbitrary* signatures. Therefore, we gave up the usual restriction of SPO-rewriting to *graph structures* which are signatures with *unary* operation symbols only, compare [13]. Categories of partial morphisms constructed over graph structures have an initial object and *all* pushouts, which means that all finite co-limits can be constructed. This property leads to a rich theory, compare again [13]. This is no longer the case if we admit constants and operation symbols with more than one argument even if we pass from total to partial algebras.

Partial algebras, however, allow a simple reduction to total graph structures which are constrained by a set of Horn-formulas. The reduction provides an easy to check and characterising condition for the existence of pushouts of partial morphisms, namely that the pushout of partial algebras coincides with the pushout constructed in the underlying graph structure.[1]

In this paper, we discuss which types of *additional* constraints on categories of partial algebras preserve this property that the operational semantics is just well-known rewriting of graph structures. We present two types of such constraints.

[1] For details compare [18].

The first type uses arbitrary conditional equations, which can for example be used to specify that some operations are injective or that different compositions of operations lead to the same result if applied to the same argument.[2] The second type of constraints discussed in this paper specifies that some unary operations shall be total, i. e. defined for all arguments. Total and injective operations can be used to model the object-oriented concept of inheritance by sub-type inclusions, compare Section 5.

The paper is structured as follows. In Section 2, we recapitulate our notion of partial algebra and the reduction to hierarchical graph structures. Section 3 presents and generalises the results of [18] concerning existence and characterisation of pushouts in categories of partial morphism over partial algebras. On the basis of these results, the main Section 4 introduces new constraints which preserve the operational semantics of SPO-rewriting for graph structures and, therefore, can be interpreted as global application conditions. Section 5 demonstrates the gain in expressive power which we obtain by using constrained partial algebras. Finally, Section 6 discusses related word and future research issues.

2 Partial Algebras and Hypergraphs

A *signature* $\Sigma = (S, O)$ consists of a set of sort names S and a domain- and co-domain-indexed family of operation names $O = (O_{w,v})_{w,v \in S^*}$. A *partial* Σ-*algebra* $A = \left(A_S, O^A\right)$ is a family $A_S = (A_s)_{s \in S}$ of carrier sets together with a partial map $o^A : A^w \to A^v$ for every operation symbol $o \in O_{w,v}$ with $w, v \in S^*$.[3]

A *homomorphism* $h : A \to B$ between two partial algebras A and B wrt. the same signature $\Sigma = (S, O)$ is a family of mappings $h = (h_s : A_s \to B_s)_{s \in S}$, such that, for all operation symbols $o \in O_{w,v}$, the following condition is satisfied: If $o^A(x)$ is defined in A for $x \in A^w$, then $o^B(h^w(x))$ is defined in B and $h^v\left(o^A(x)\right) = o^B\left(h^w(x)\right)$.[4] A homomorphism $h = (h_s : A_s \to B_s)_{s \in S}$ is *closed*, if we have for every operation $o \in O_{w,v}$: Whenever o^B is defined for $h^w(x)$ there is $x' \in A^w$ with $h^w(x) = h^w(x')$ and o^A is defined for x'.[5]

The category of all partial Σ-algebras and all homomorphisms between them is denoted by $\mathcal{A}_\Sigma$. By $\underline{\mathcal{A}_\Sigma}$ we denote the subcategory of all total Σ-algebras. Note that all homomorphisms in $\underline{\mathcal{A}_\Sigma}$ are closed.

A (constructive) Σ-*contraint* is given by an epimorphism $c : P \twoheadrightarrow C$ from a Σ-algebra P, called the *premise*, to a Σ-algebra C which is called the *conclusion*.

[2] Commutativity in the categorical sense.

[3] By this definition, an operation symbol $o \in O_{w,\epsilon}$ is interpreted in an algebra A as a partial operation into an one-element-set, i. e. $o^A : A^w \to \{*\}$, which means that o^A singles out a sub-set of A^w only, namely the sub-set where it is defined. Hence, o^A is a *predicate*.

[4] Given a sort indexed family of mappings $(f_s : G_s \to H_s)_{s \in S}$, $f^w : G^w \to H^w$ is recursively defined for every $w \in S^*$ by (i) $f^\epsilon = \{(*, *)\}$, (ii) $f^w = f_s$ if $w = s \in S$, and (iii) $f^w = f^v \times f^u$, if $w = vu$.

[5] This means that definedness in B stems from definedness in A.

A homomorphism $h : P \to A$ *solves* the constraint $c : P \twoheadrightarrow C$, written $h \models c$, if there is homomorphism $h^* : C \to A$ such that $h^* \circ c = h$. An algebra A *satisfies* a constraint $c : P \twoheadrightarrow C$, written $A \models c$, if every morphism $h : P \to A$ solves c. Given a set $\mathfrak{C}$ of Σ-constraints, $\mathcal{A}_{\Sigma,\mathfrak{C}}$ denotes the full sub-category of $\mathcal{A}_\Sigma$ of all algebras that satisfy all the constraints $c \in \mathfrak{C}$. Such a category specified by Σ-constraints is called a *quasi-variety.* It is well-known that a full sub-category of $\mathcal{A}_\Sigma$ is a quasi-variety, if and only if it is an epi-reflective sub-category of $\mathcal{A}_\Sigma$.[6] Typically, a constraint is syntactically presented as an implication from a syntactical presentation of the premise to a syntactical presentation of the conclusion.

By contrast to total algebras, epimorphisms in categories of partial algebras need not be surjective in each component.[7] Thus, constraints can express some definedness requirements. Consider as an example the (unconditional) clause $\mathtt{x} \in \mathtt{S} : \mathtt{f}(\mathtt{f}(\mathtt{x})) = \mathtt{f}(\mathtt{x})$ for a signature Σ with an operation symbol $\mathtt{f} : \mathtt{S} \to \mathtt{S}$. It requires `f` to be idem-potent *and total.*

Given a signature $\Sigma = (S, O)$, $\Sigma^{\mathrm{u}} = (S^{\mathrm{u}}, O^{\mathrm{u}})$ denotes the underlying graph structure which is defined on sorts by:

$$S^{\mathrm{u}} = S \uplus \biguplus_{w,v \in S^*} O_{w,v}.$$

For every operation symbol $o \in O_{s_1 \dots s_j\,,\, s_{j+1} \dots s_{j+k}}$ in Σ with $j, k \geq 0$, O^{u}_{o,s_i} contains an operation symbol d^o_i for $1 \leq i \leq j$ and $O^{\mathrm{u}}_{o,s_{j+i}}$ contains an operation symbol c^o_i for $1 \leq i \leq k$.[8] There are no other operation symbols in O^{u}.

Note that the signature $\Sigma^{\mathrm{u}} = (S^{\mathrm{u}}, O^{\mathrm{u}})$ constitutes a hierarchical graph structure in the sense of [13].[9] In S^{u}, the sorts in S are on level 0 and the sorts in $\biguplus_{w,v \in S^*} O_{w,v}$ are on level 1. All operations in O^{u} are unary and map from sorts on level 1 to sorts on level 0. Thus, a total algebra wrt. $\Sigma^{\mathrm{u}} = (S^{\mathrm{u}}, O^{\mathrm{u}})$ can be interpreted as a hypergraph having vertices typed in S^{u} and hyperedges typed in O^{u}.

Let $\mathcal{G}_\Sigma$ denote the category of all *total* Σ^{u}-algebras and Σ^{u}-homomorphisms, i. e. $\mathcal{G}_\Sigma$ is short for $\underline{\mathcal{A}}_{\Sigma^{\mathrm{u}}}$.[10] Then there is a full and faithful functor $\gamma : \mathcal{A}_\Sigma \to \mathcal{G}_\Sigma$ mapping each partial algebra $A \in \mathcal{A}_\Sigma$ to $\gamma(A) \in \mathcal{G}_\Sigma$ by setting

1. for each sort $s \in S$: $\gamma(A)_s = A_s$ and

[6] A category $\mathcal{S}$ is an epi-reflective sub-category of a category $\mathcal{C}$, if it is a sub-category of $\mathcal{C}$, i. e. $\mathcal{S} \subseteq \mathcal{C}$, and for every object $C \in \mathcal{C}$ there is a $\mathcal{C}$-epi-morphism $\eta_C : C \twoheadrightarrow S$ such that $S \in \mathcal{S}$ and for every morphism $f : C \to S'$ with $S' \in \mathcal{S}$, there is a unique morphism $f^* : S \to S'$ with $f^* \circ \eta_C = f$. For the results about epi-reflection, compare [21] for the total case and [1] for the partial case. They can also be found in [14].

[7] Compare [1,14].

[8] d^o_i and c^o_i are short for i-th domain respectively co-domain of operation o.

[9] A signature $\Sigma = (S, O)$ is a *graph structure*, if it contains unary operation symbols only, i. e. $O_{w,v} = \emptyset$, if $|w| = 0$ or $|w| \geq 2$. It is *hierarchical*, if there is no family $(o_i \in O_{s_i,v_i})_{i \in \mathbb{N}}$, such that, for all $i \in \mathbb{N}$, $v_i = x_i s_{i+1} y_i$ with $x_i, y_i \in S^*$.

[10] It is well-known that $\mathcal{G}_\Sigma$ is complete and co-complete.

2. for each operation $o \in O_{s_1 \ldots s_j\ ,\, s_{j+1} \ldots s_{j+k}}$ with $i, k \geq 0$:
   (a) $\gamma(A)_o = o^A$ and
   (b) for all $(x, y) = ((x_1, \ldots, x_j), (y_1, \ldots, y_k)) \in o^A$, $1 \leq m \leq j$, $1 \leq n \leq k$:[11]
      i. $(\mathrm{d}^o_m)^{\gamma(A)}(x, y) = x_m$ and
      ii. $(\mathrm{c}^o_n)^{\gamma(A)}(x, y) = y_n$

and each homomorphism $h : A \to B$ in $\mathcal{A}_\Sigma$ to $\gamma(h) : \gamma(A) \to \gamma(B)$ by setting

1. for each sort $s \in S$: $\gamma(h)_s = h_s$ and
2. for each operation $o \in O_{w,v}$ with $w, v \in S^*$: $\gamma(h)_o(x, y) = (h^w(x), h^v(y))$.

Proposition 1 (Preservation of Epimorphisms). *If $\gamma : \mathcal{A}_\Sigma \to \mathcal{G}_\Sigma$ is the full and faithful functor from partial Σ-algebras to the underlying hierarchical Σ^{u}-graph-structures, then $\gamma(h) : \gamma(A) \to \gamma(B)$ is epimorphism, if and only if $h : A \to B$ is a closed epimorphism.*

Proof. "$\Leftarrow$": Every closed epimorphism is surjective on all sorts and, by definition of closedness, also on "operations". "$\Rightarrow$": Suppose $\gamma(h) : \gamma(A) \to \gamma(B)$ is epimorphism, i. e. is surjective in each component. Then $h_s = \gamma(h)_s$ is surjective for each sort s. Thus, h is epimorphism. If o^B is defined for $h^w(x)$, $(h^w(x), y) \in o^B$. Since $\gamma(h)_o$ is surjective for every operation o, $(h^w(x), y) = \gamma(h)_o(x', y')$ which means that $h^w(x) = h^w(x')$ and o^A is defined for x'. Therefore, h is closed. $\square$

Unfortunately, the functor γ is not *isomorphism-dense*[12], such that $\mathcal{A}_\Sigma$ and $\mathcal{G}_\Sigma$ are not equivalent. We have to further restrict $\mathcal{G}_\Sigma$ by the following family of constraints which formalises uniqueness of partial maps:

$$\mathcal{U} = \Big(\forall e_1, e_2 \in o : (\mathrm{d}^o_i(e_1) = \mathrm{d}^o_i(e_2))_{1 \leq i \leq |w|} \implies e_1 = e_2\Big)_{w,v \in S^*, o \in O_{w,v}} \tag{1}$$

By $C_\mathcal{U}$, we denote the constraints (epimorphisms) presented by the implications $\mathcal{U}$. Fig. 1 illustrates the correspondence between $\mathcal{U}$ and $C_\mathcal{U}$ for the sample signature $\Sigma_{\mathtt{f2}}$. Since there is only one operation, namely $\mathtt{f} : \mathtt{S}, \mathtt{S} \to \mathtt{S}$, $C_\mathcal{U}$ contains a single epimorphism only, which is depicted in Figure 1 by the dotted arrows.[13]

In the following, $\mathcal{P}_\Sigma \subseteq \mathcal{G}_\Sigma$ denotes the quasi-variety of all algebras in $\mathcal{G}_\Sigma$ satisfying all the constraints in $\mathcal{U}$, i. e. $\mathcal{P}_\Sigma = \underline{\mathcal{A}}_{\Sigma^{\mathrm{u}}, C_\mathcal{U}}$. Since $\mathcal{P}_\Sigma$ is an epi-reflection of the category $\mathcal{G}_\Sigma$, we obtain a pair $(F(A) \in \mathcal{P}_\Sigma, \eta_A : A \twoheadrightarrow F(A))$ for every $A \in \mathcal{G}_\Sigma$ such that for every other pair $(X \in \mathcal{P}_\Sigma, f : A \to X)$, there is a unique $f^* : F(A) \to X$ with $f^* \circ \eta_A = f$. Since every image of γ is in $\mathcal{P}_\Sigma$ and γ is isomorphism-dense wrt. $\mathcal{P}_\Sigma$, we have:

[11] Note that all operations in $\gamma(A)$ are just projections!

[12] A functor $\gamma : \mathcal{A} \to \mathcal{B}$ between categories $\mathcal{A}$ and $\mathcal{B}$ is isomorphism-dense, if for every $B \in \mathcal{B}$ there is $A \in \mathcal{A}$ such that $\gamma(A) \cong B$.

[13] Note that this visualisation suggests that the constraints in $\mathcal{U}$ can be interpreted as total and non-injective SPO graph rewriting rules. Indeed, applying these rules until every further application leads to an identity trace, provides a constructive way to "execute" the reflection from $\mathcal{G}_\Sigma$ to $\mathcal{A}_\Sigma$.

```
Σf2 =
sorts S
opns  f: S,S --> S
```

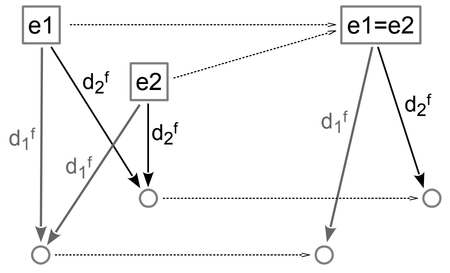

Figure 1. Example for a Uniqueness Constraint

Fact 2 (Partial Algebras as Hypergraphs). *$\mathcal{A}_\Sigma$ and $\mathcal{P}_\Sigma$ are equivalent.*

Therefore, partial algebras can be considered as special hypergraphs which do not allow multiple edges of the same type between the same domain vertices.

Since $\mathcal{A}_\Sigma$ and $\mathcal{P}_\Sigma$ are equivalent, all results we obtain for $\mathcal{P}_\Sigma$ in the following are also valid in $\mathcal{A}_\Sigma$. Since $\mathcal{P}_\Sigma$ is an epi-reflection of $\mathcal{G}_\Sigma$, $\mathcal{P}_\Sigma$ is closed wrt. sub-objects and products. Thus, pullbacks in $\mathcal{P}_\Sigma$ coincide with pullbacks in $\mathcal{G}_\Sigma$. Pushouts in $\mathcal{P}_\Sigma$ are quotients of pushouts in $\mathcal{G}_\Sigma$ in the following sense: The pushout of $(f : A \to B, g : A \to C)$ in $\mathcal{P}_\Sigma$ is given by $(\eta_D \circ g^* : B \to D', \eta_D \circ f^* : C \to D')$ where $(g^* : B \to D, f^* : C \to D)$ is the pushout of f and g in $\mathcal{G}_\Sigma$ and $\eta_D : D \twoheadrightarrow D'$ is the epi-reflector that transfers the $\mathcal{G}_\Sigma$-object D into $\mathcal{P}_\Sigma$.

3 Partial Morphisms for Algebras and Hypergraphs

The single-pushout approach to rewriting uses partial morphisms as rules, total morphisms as matches, and pushouts in *categories of partial morphisms* as direct derivations. Therefore, we have to proceed from the categories $\mathcal{G}_\Sigma$ and $\mathcal{P}_\Sigma$ with total morphisms to the categories $\mathcal{G}^{\mathrm{P}}_\Sigma$ and $\mathcal{P}^{\mathrm{P}}_\Sigma$ of hypergraphs and partial algebras with partial morphisms.

A *concrete partial morphism* in $\mathcal{G}^{\mathrm{P}}_\Sigma$ is a span of $\mathcal{G}_\Sigma$-morphisms $(p : K \rightarrowtail P, q : K \to Q)$ such that p is monic. Two concrete partial morphisms (p_1, q_1) and (p_2, q_2) are equivalent and denote the same *abstract partial morphism* if there is an isomorphism i such that $p_1 \circ i = p_2$ and $q_1 \circ i = q_2$; in this case we write $(p_1, q_1) \equiv (p_2, q_2)$ and $[(p, q)]_\equiv$ for the class of spans that are equivalent to (p, q). The *category of partial morphisms* $\mathcal{G}^{\mathrm{P}}_\Sigma$ over $\mathcal{G}_\Sigma$ has the same objects as $\mathcal{G}_\Sigma$ and abstract partial morphisms as arrows. The identities are defined by $\mathrm{id}_A^{\mathcal{G}^{\mathrm{P}}_\Sigma} = \left[(\mathrm{id}_A^{\mathcal{G}_\Sigma}, \mathrm{id}_A^{\mathcal{G}_\Sigma})\right]_\equiv$ and composition of partial morphisms $[(p : K \rightarrowtail P, q : K \to Q)]_\equiv$ and $[(r : J \rightarrowtail Q, s : J \to R)]_\equiv$ is given by

$$[(r, s)]_\equiv \circ_{\mathcal{G}^{\mathrm{P}}_\Sigma} [(p, q)]_\equiv = [(p \circ_{\mathcal{G}_\Sigma} r' : M \rightarrowtail P, s \circ_{\mathcal{G}_\Sigma} q' : M \to R)]_\equiv$$

where $(M, r' : M \rightarrowtail K, q' : M \to J)$ is pullback of q and r. Note that there is the faithful embedding functor $\iota : \mathcal{G}_\Sigma \to \mathcal{G}^{\mathrm{P}}_\Sigma$ defined by identity on objects and

$$\begin{tikzcd}
L & K \arrow[l, hook', "l"'] \arrow[r, "r"] & R \\
P \arrow[u, hook, "p"] \arrow[d, "q"'] \arrow[ur, phantom, "(1)"] & D \arrow[l, hook', "\overline{l}"'] \arrow[r, "\overline{r}"] \arrow[u, hook, "\overline{p}"] \arrow[d, "\overline{q}"] \arrow[ur, phantom, "(2)"] \arrow[dl, phantom, "(3)"] & P^* \arrow[u, hook, "p^*"'] \arrow[d, "q^*"] \arrow[dl, phantom, "(4)"] \\
Q & K^* \arrow[l, hook', "l^*"] \arrow[r, "r^*"'] & H
\end{tikzcd}$$

Figure 2. Pushout in $\mathcal{G}_\Sigma^{\mathrm{P}}$

$(f : A \to B) \mapsto [\mathrm{id}_A : A \rightarrowtail A, f : A \to B]$ on morphisms. We call $[d : A' \rightarrowtail A, f : A' \to B]$ a *total* morphism and, by a slight abuse of notation, write $[d, f] \in \mathcal{G}_\Sigma$, if d is an isomorphism. From now on, we mean the abstract partial morphism $[f, g]_\equiv$ if we write $(f : B \rightarrowtail A, g : B \to C)$. If the monic component in a partial morphism is an inclusion, we also write $g : A \overset{B}{\dashrightarrow} C$ for $(f : B \hookrightarrow A, g : B \to C)$. We omit the annotation of the arrow, if the sub-object of the partial morphism is irrelevant or uniquely determined by some universal properties.
It is well-known that $\mathcal{G}_\Sigma^{\mathrm{P}}$ is complete and co-complete.[14]

Construction 3 (Pushout in $\mathcal{G}_\Sigma^{\mathrm{P}}$). For partial morphisms $r : L \overset{K}{\dashrightarrow} R$ and $q : L \overset{P}{\dashrightarrow} Q$, the pushout morphisms $r^* : Q \overset{K^*}{\dashrightarrow} H$ and $q^* : R \overset{P^*}{\dashrightarrow} H$ are constructed as follows, compare Figure 2:

1. D is the largest sub-algebra in $K \cap P$ satisfying:
   (a) $r(x) = r(y) \wedge x \in D \implies y \in D$ and
   (b) $q(x) = q(y) \wedge x \in D \implies y \in D$.
2. $\overline{l} : D \hookrightarrow P$ and $\overline{p} : D \hookrightarrow K$ are the corresponding inclusions.
3. P^* and K^* are the largest sub-algebras in $R - r(K - D)$ resp. $Q - q(P - D)$.
4. $p^* : P^* \hookrightarrow R$ and $l^* : K^* \hookrightarrow Q$ are the corresponding inclusions.
5. $\overline{r} : D \to P^*$ is defined by $d \mapsto r(d)$ and $\overline{q} : D \to K^*$ by $d \mapsto q(d)$.
6. (q^*, r^*) is the pushout of $(\overline{q}, \overline{r})$ in $\mathcal{G}_\Sigma$.

*Remark*s. Construction 3 leads to the four commutative squares (1) – (4) in Figure 2. They possess the following properties:

1. Squares (2) and (3) are pullbacks such that $r^* \circ q = q^* \circ r$.
2. Squares (1) – (3) make up a final triple in the sense of [19].
3. Square (4) is hereditary pushout in $\mathcal{G}_\Sigma$ since all pushouts in $\mathcal{G}_\Sigma$ are hereditary, compare Definition 4 below.

Therefore, construction 3 provides a pushout in $\mathcal{G}_\Sigma^{\mathrm{P}}$ due to the following general fact: A diagram as in Figure 2 is pushout of partial morphisms over an arbitrary category $\mathcal{C}$, if and only if (1) – (3) make up a final triple in $\mathcal{C}$ and (4) is hereditary pushout in $\mathcal{C}$, compare [19,20].

[14] Compare for example [13]

Definition 4 (Hereditary Pushout). *A pushout (p', q') of (p, q) in an arbitrary category is* hereditary, *if for each commutative cube as in Fig. 3, which has pullback squares (q_i, i_0) and (p_i, i_0) of (i_1, q) and (i_2, p) resp. as back faces with monic i_1 and i_2 the following compatibility between pushouts and pullbacks holds: In the top square, (q'_i, p'_i) is pushout of (p_i, q_i), if and only if, in the front faces, (p'_i, i_1) and (q'_i, i_2) are pullbacks of (i_3, p') and (i_3, q') resp. and i_3 is monic.*[15]

$$\begin{tikzcd}
& A' \arrow[rr, "q_i"] \arrow[dl, "p_i"'] \arrow[dd, tail, "i_0"' near start] & & C' \arrow[dl, "p'_i"] \arrow[dd, tail, "i_1"] \\
B' \arrow[rr, "q'_i"' near start] \arrow[dd, tail, "i_2"'] & & D' \arrow[dd, tail, "i_3" near end] & \\
& A \arrow[rr, "q" near start] \arrow[dl, "p"'] & & C \arrow[dl, "p'"] \\
B \arrow[rr, "q'"'] & & D &
\end{tikzcd}$$

Figure 3. Hereditary Pushout

$\mathcal{P}^{\mathrm{P}}_{\Sigma}$ is the full sub-category of $\mathcal{G}^{\mathrm{P}}_{\Sigma}$ determined by the object inclusion of $\mathcal{P}_{\Sigma} \subseteq \mathcal{G}_{\Sigma}$. $\mathcal{P}^{\mathrm{P}}_{\Sigma}$ does not possess all pushouts. $\mathcal{P}_{\Sigma}$ has all final triples but not all pushouts in $\mathcal{P}_{\Sigma}$ are hereditary. Final triples are constructed as in $\mathcal{G}_{\Sigma}$, since steps (1) – (5) of Construction 3 produce sub-objects D, P^*, and K^* which satisfy all Horn-formulae in $\mathcal{U}$ (compare (1) on page 4), if L, R, and Q do. Hereditariness of pushouts in $\mathcal{P}_{\Sigma}$ is characterised by an easy to check condition:

Fact 5 (Hereditrary Pushouts in $\mathcal{P}_{\Sigma}$). *A pushout in $\mathcal{P}_{\Sigma}$ is hereditary, if and only if it is pushout in $\mathcal{G}_{\Sigma}$.*

The proofs for this fact are provided by the proofs for Proposition 7 ($\Leftarrow$) and Proposition 8 ($\Rightarrow$) in [18]. Since all arguments in these proofs do not refer to the concrete structure of the formulae in $\mathcal{U}$, the result can be generalised as follows:

Theorem 6 (Hereditary Pushouts in Reflective Sub-Categories of $\mathcal{G}_{\Sigma}$). *For every epi-reflective sub-category $\mathcal{C}_{\Sigma}$ of $\mathcal{G}_{\Sigma}$, we have:*

1. *Pushouts in $\mathcal{C}_{\Sigma}$ are hereditary, if and only if they are pushouts in $\mathcal{G}_{\Sigma}$.*
2. *Let $\mathcal{C}^{\mathrm{P}}_{\Sigma}$ be the full sub-category of $\mathcal{G}^{\mathrm{P}}_{\Sigma}$ determined by the object inclusion of $\mathcal{C}_{\Sigma} \subseteq \mathcal{G}_{\Sigma}$: If a pushout for a pair of morphisms in $\mathcal{C}^{\mathrm{P}}_{\Sigma}$ exists, then it coincides with the pushout of the pair constructed in $\mathcal{G}^{\mathrm{P}}_{\Sigma}$.*

[15] For details on hereditary pushouts see [11,12]

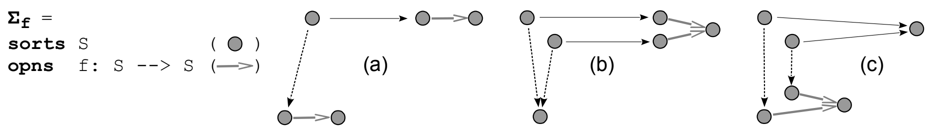
```
Σf =
sorts S              ( ● )
opns  f: S --> S  (—>)
```

Figure 4. Examples for Impossible Rewrites

4 SPO-Rewriting of Constrained Partial Algebras

Fact 5 shows that SPO-rewriting in $\mathcal{P}^{\mathrm{P}}_{\Sigma}$ is just SPO-rewriting in graph structures from the operational point of view. It only adds an application condition, namely that rewriting a partial algebra (as a graph structure) must result in a graph structure that represents a partial algebra.

Figure 4 presents three typical impossible rewrites in $\mathcal{P}^{\mathrm{P}}_{\Sigma_f}$. The mappings of the rule morphisms, which are all total in the three examples, are depicted by black straight arrows and the mappings of the match morphisms by dotted arrows. In the situation (a), the rule tries to add a definition of `f` to an object in the host graph that possesses a definition of `f` already. Situation (b) wants to add a new object as the result of the application of `f` for two existing objects. The match identifies the two existing objects. Rewriting in the underlying graph structure produces two parallel "edges" between the old and the new object which does not satisfy the uniqueness condition $\mathcal{U}$, compare (1) on page 4. Situation (c) is kind of symmetric to situation (b). Here two existing objects have the same result under application of `f`. The rule tries to merge these two objects, which again leads to two parallel "edges" violating $\mathcal{U}$.

If we add more constraints on partial algebras that are subject to SPO-rewriting, we want to preserve this fundamental property, i. e.:

Operational Semantics *Every SPO-rewrite of a constrained partial algebra coincides with the SPO-transformation of the underlying graph structure.*

Theorem 6 provides a first sort of constraints that satisfy this criteria: namely arbitrary constraints wrt. $\mathcal{G}_{\Sigma}$.[16] But these constraints do not one-to-one correspond to general constraints on $\mathcal{P}_{\Sigma}$. Constraints on partial algebras can formulate conditional equalities *and* definedness requirements, since epimorphisms in categories of partial algebras need not be surjective. Therefore, we have to restrict the constraints to those that do not implicitly formulate definedness requirements. By Proposition 1, these constraints are exactly characterised by the epimorphism that are closed. We call a constraint of this type *conditional equation*, since its conclusion in the syntactical presentation as an implication consists of equalities between variables only. Note that the set of constraints presented by $\mathcal{U}$ on page 4 is a set of conditional equations.

[16] Recall that $\mathcal{G}_{\Sigma}$ is the underlying graph structure of $\mathcal{P}_{\Sigma}$.

```
Spec_cycle =
sorts  S,S'          (●,○)
opns   f: S --> S  (——>)
       i: S --> S' (----->)
axms   i(x)=i(y)==> x=y
```

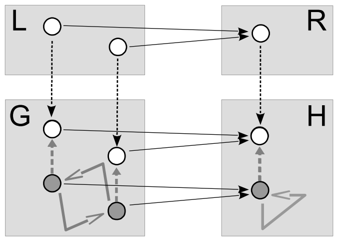

Figure 5. Operation Cycles

Corollary 7 (Conditional Equation). *If $\mathcal{A}_\Sigma$ is a category of partial algebras and E a set of conditional equations wrt. Σ, then every pushout in $\mathcal{A}^{\mathrm{P}}_{\Sigma,E}$ coincides with the pushout constructed in $\mathcal{G}^{\mathrm{P}}_\Sigma$.*

Conditional equations provide a rich supply for useful constraints on partial algebras which are subject to rewriting, for example:

Singleton: $x = x'$,
Injectivity: $f(x) = f(x') \implies x = x'$,
Joint-Injectivity: $f_1(x) = f_1(x'), \ldots, f_n(x) = f_n(x') \implies x = x'$,
Commutativity: $f(g(x)) = z, h(x) = z' \implies z = z'$,
Inverse: $f(x) = z, g(z) = x' \implies x = x'$, and
Mutual Inverse: $f(x) = z, g(z) = x', f(x') = z' \implies x = x', z = z'$.

Note that the the last five examples require equalities only in situations where the operations are "defined enough", i. e. the premises can be satisfied.

Up to this point, we do not have any means to require some sort of definedness. As it has been shown in [13], requiring definedness of operation symbols having more than one argument or none can lead to situations in which a final triple does not exist, compare steps (1) – (5) of Construction 3. Even worse: If operations with at least two arguments are required to be defined for a certain range of arguments, there are no obvious conditions which decide whether or not the final triple in Construction 3 exists.

Therefore, we restrict definedness requirements to unary operations here as well. Unfortunately, we cannot be as liberal as we want to at this point. This is due to the fact that even definedness requirements for unary operations may heavily interact with other constraints in form of conditional equations in an undesirable way.

Example 8. [Interference of Equations and Definedness]A good example for such an interference is depicted in Figure 5. The shown specification requires its two unary operations f and i to be total, i. e. to be defined for all arguments. We – here and in the following – indicate this requirement by underlining the affected operations. Additionally, the operation i is forced to be injective.

The right part of the figure shows a pushout of two morphisms from L to R and L to G indicated by the black straight and dotted arrows resp. Note that the pushout does not coincide with the pushout constructed in the underlying graph structure. But it is hereditary.

This is mainly due to the fact, that G admits five sub-algebras only, namely the empty graph, G itself, and the three possible sub-graphs consisting of elements of the sort $\mathtt{S}'$ only. Only three of them can occur in a commutative cube as in Figure 3, namely the empty graph, G, and the sub-graph consisting of the two $\mathtt{S}'$-elements. And all these three cubes satisfy the hereditariness requirement. □

Example 8 shows a situation that violates our requirement "Operational Semantics" on page 8: We have hereditariness without coincidence of the pushout in the constrained and the unconstrained category. This is mainly due to the cyclic structure induced by the operation $\underline{\mathtt{f}}$. Therefore, we restrict definedness requirements of unary operations to those that do not lead to cycles, i. e. to those that produce a hierarchical underlying graph structure.

Definition 9 (Constrained Category of Partial Algebras). *A* constrained category $\mathcal{R}_{(\Sigma,C,T)}$ of partial algebras *is syntactically given by a triple* (Σ, C, T)*, where* $\Sigma = (S, O)$ *is a signature,* C *is a set of conditional equations and* $T = (T_{s,w} \subseteq O_{s,w})_{s\in S, w\in S^*}$ *is a sub-set of the unary operations in* O *satisfying the following hierarchy condition: There is no family* $(o_i \in T_{s_i,w_i})_{i\in\mathbb{N}}$*, such that, for all* $i \in \mathbb{N}$*,* $w_i = v_i s_{i+1} u_i$ *with* $v_i, u_i \in S^*$*. The semantics of* $\mathcal{R}_{(\Sigma,C,T)}$ *is the full sub-category of* $\mathcal{A}^{\mathrm{P}}_{\Sigma,E}$ *of those algebras in which all operations in* T *are total.*

As a generalisation of Propositions 7 and 8 in [18], we obtain:

Theorem 10 (Hereditariness in Constrained Categories). *A pushout in* $\mathcal{R}_{(\Sigma,C,T)}$ *is hereditary, if and only if it is pushout in the underlying graph structure* $\mathcal{G}_\Sigma$*.*

Proof. "$\Leftarrow$": In addition to the arguments in the proof of Proposition 7 in [18], we have to show that a pushout $(f^* : C \to D, g^* : B \to D)$ in $\mathcal{G}_\Sigma$ for $(f : A \to B, g : A \to C)$ satisfies the definedness requirements specified by T. So let $o \in T_{s,w}$ and $x \in D_s$. Then $x = f^*(x_c)$ or $x = g^*(x_b)$. Without loss of generality assume the first. Since C satisfies the definedness requirements, $o^C(x_c)$ is defined. Since $(f^*)^w\left(o^C(x_c)\right) = o^P(f^*_s(x_c)) = o^P(x)$, o^P is defined for x.

"$\Rightarrow$" (Sketch): Here, we can repeat the arguments in the proof of Proposition 8 in [18]. Let $(f : A \to B, g : A \to C)$ be given, $(f^* : C \to D, g^* : B \to D)$ the pushout in $\mathcal{G}_\Sigma$, and $(f' : C \to E, g' : B \to E)$ the pushout in $\mathcal{R}_{(\Sigma,C,T)}$. Since D satisfies all definedness requirements (see above), E is a quotient of D and the two pushout morphisms (f', g') are jointly surjective. If the two pushouts are different, there are two elements $x, y \in B \uplus C$ which are mapped to the same element z by f' and/or g' in E and to different elements z_1, z_2 by f^* and/or g^* in D. Now, we construct sub-algebras of $A' \subseteq_A A$, $B' \subseteq_B B$, and $C' \subseteq_C C$ by erasing x and recursively its minimal context in all three algebras such that $(\subseteq_A$

$, f_{|A'})$ becomes the pullback of $(\subseteq_B, f)$ and $(\subseteq_A, g_{|A'})$ becomes the pullback of $(\subseteq_C, g)$. Since all total operations are hierarchical and x and y are not identified by f^* and g^*, the element y remains in B' or C' and we obtain $z' = f'_{|A'}(y)$ or $z' = g'_{|A'}(y)$ as an element in the pushout $(E', f'_{|A'} : C' \to E', g'_{|A'} : B' \to E'$ of $(f_{|A'}, g_{|B'})$. The universal morphism u from E' to E maps z' to z. Now, we have $u(z') = z$ and $f'(x) = z'$ or $g'(x) = z'$. Since by construction, x is neither in B' nor in C', either $(\subseteq_B, g'_{|A'})$ is not pullback of (u, g') or $(\subseteq_C, f'_{|A'})$ is not pullback of (u, f'). □

Having this result about hereditary pushouts, we can conclude:

Corollary 11 (Rewriting in Constrained Categories). *Every pushout in the category $\mathcal{R}^{\mathrm{P}}_{(\Sigma,C,T)}$ of partial morphisms wrt. a constrained category $\mathcal{R}_{(\Sigma,C,T)}$ coincides with the pushout constructed in the underlying graph structure $\mathcal{G}^{\mathrm{P}}_{\Sigma}$.*

Remark. Note that step (1) in Construction 3 on page 6 needs a marginal modification. Now we construct D as the largest sub-algebra *satisfying all definedness constraints in T* with properties (1a) and (1b). This algebra always exists, since all total operations are unary and the construction ends up in the standard cascade-on-delete behaviour that is well-known from single-pushout rewriting.[17]

The results of this section offer better ways to specify the structures that are subject to single-pushout rewriting. The appropriate framework is provided by the notion of *constrained category of partial algebras*, compare Definition 10. By Corollary 11, a simple operational semantics for rewrites in constrained categories is guaranteed. Therefore, there is a good chance to transfer some SPO-theory from graph structures to partial algebras and even to constrained partial algebras. This is subject of future research. In the rest of the paper, we want to present the gain of expressive power for SPO-rewriting of constrained structures.

5 Inheritance – the Algebraic Way

SPO-rewriting in categories of constrained partial algebras provides mechanisms to model many object-oriented concepts incl. inheritance in a straightforward and appropriate way. The inheritance model which we introduce below goes beyond all existing ones,[18] since it provides easy means for "runtime" type-specialisation and -generalisation of objects. Here is the general recipe how object-oriented class and type models are translated into constrained categories of partial algebras:

[17] Cacade-on-delete is a notion well-known from relational databases: If a row in a relation is deleted, all rows pointing directly or indirectly to it by foreign keys are deleted as well. SPO-rewriting provides exactly the same effect: If a vertex is deleted all edges adjacent to the vertex are deleted as well. In arbitrary graph structure this effect can cascade as well, if we have more than 2 hierarchy levels, see [13] for details.

[18] Compare [4,8,9,17] for the double-pushout, [6,16] for the single-pushout, and [15] for the sesqui-pushout approach.

Immutable base types, like `Integer` and `String`, and the operations on these base types, like concatenation (`+`) of strings and addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`) of integers are just modelled as a part of the underlying signature by appropriate sort and operation symbols. On the object-level, i. e. on the level of the objects that are subject to SPO-transformation these sort and operation symbols are interpreted by the standard carrier sets and partial operations. Note that the interpretation as *partial* operations provides an adequate model for `overflow`-situations or `division-by-zero`. In transformation rules, partial term algebras over a suitable set of variables are used. By contrast to total term algebras which are almost always infinite, the partial term algebras used in rules can always be chosen as finite algebras.[19]

Classes are modelled by sorts. So all types (base or not) are modelled by sorts.

An *attribute* $\mathtt{a}$ of base type $\mathtt{T}$ in a class $\mathtt{C}$ is modelled (i) by a partial operation $\mathtt{a} : \mathtt{C} \to \mathtt{T}$, if its multiplicity is zero or one (`0..1`), (ii) a total operation $\underline{\mathtt{a}} : \mathtt{C} \to \mathtt{T}$,[20] if its multiplicity is exactly one (`1`) (iii) by a predicate $\mathtt{a} : \mathtt{C}, \mathtt{T} \to \{*\}$, if its multiplicity is many incl. none (`*`) without double assignments of the same value to the same object,[21] and (iii) by an additional sort symbol $\mathtt{A}$ and *total* operations $\underline{\mathtt{a_o}} : \mathtt{A} \to \mathtt{C}$ and $\underline{\mathtt{a_t}} : \mathtt{A} \to \mathtt{T}$, if its multiplicity is many incl. none (`*`) and multiple assignments of the same value to the same object are allowed.[22]

An *association* $\mathtt{r}$ between classes $\mathtt{C}$ and $\mathtt{D}$ is modelled as follows: If the multiplicity of $\mathtt{r}$ at both ends is many incl. none (`*`), we devise a predicate $\mathtt{r} : \mathtt{C}, \mathtt{D} \to \{*\}$ if multiple links between the same pair of objects are forbidden or we add an additional sort $\mathtt{R}$ and two *total* operations $\underline{\mathtt{r_C}} : \mathtt{R} \to \mathtt{C}$ and $\underline{\mathtt{r_D}} : \mathtt{R} \to \mathtt{D}$ otherwise. If the multiplicity at the $\mathtt{D}$-end is zero or one (`0..1`), we provide a partial map $\mathtt{r} : \mathtt{C} \to \mathtt{D}$ which obtains an additional injectivity constraint of the form $\mathtt{c} \in \mathtt{C} : \mathtt{r}(\mathtt{c}) = \mathtt{r}(\mathtt{c}') \implies \mathtt{c} = \mathtt{c}'$, if the $\mathtt{C}$-end has multiplicity `0..1` as well. If the multiplicity at the $\mathtt{D}$-end is exactly one (`1`), we can devise a total operation $\underline{\mathtt{r}} : \mathtt{C} \to \mathtt{D}$, as long as the signature remains hierarchical.

A *qualified association* $\mathtt{q}$ between classes $\mathtt{C}$ and $\mathtt{D}$ using a $\mathtt{T}$-typed attribute or association $\mathtt{a}$ of $\mathtt{D}$ as "key" is modelled by a partial operation $\mathtt{q} : \mathtt{C}, \mathtt{T} \to \mathtt{D}$.

If a set $\mathtt{a_1}, \ldots, \mathtt{a_n}$ of `0..1`-attributes and/or associations of the same class $\mathtt{C}$ is a key to the objects of $\mathtt{C}$, we add an injectivity constraint of the following form:

$$\mathtt{c}, \mathtt{c}' \in \mathtt{C} : \mathtt{a_1}(\mathtt{c}) = \mathtt{a_1}(\mathtt{c}'), \ldots, \mathtt{a_n}(\mathtt{c}) = \mathtt{a_n}(\mathtt{c}') \implies \mathtt{c} = \mathtt{c}'$$

Inheritance $\mathtt{S} \multimap \mathtt{G}$ of sub-class $\mathtt{S}$ from super-class $\mathtt{G}$, is modelled by a total and injective operation $\underline{\mathtt{i_{S,G}}} : \mathtt{S} \to \mathtt{G}$ and for every `diamond`-situation, defined by $\mathtt{S} \multimap \mathtt{G_1}$,

[19] For example, the total term algebra is infinite, if we have a single sort (`Nat`) with a constant (`zero`) and an unary operation (`successor`). In partial algebras, the constant need not be defined and the unary operation need not be defined everywhere. Thus, we can define them as far as they are used in the rule (e. g. to denote constants) and leave them undefined for all values that are not mentioned in the rule.

[20] Note that we indicate the constraint that an operation is total by underlining the operation name.

[21] Set semantics.

[22] Multi-set semantics.

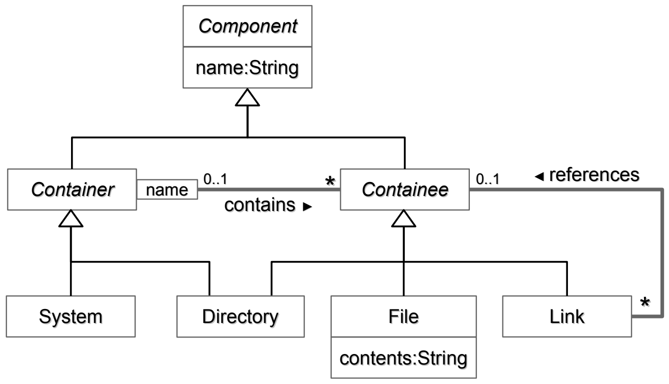

Figure 6. Class Model for File Systems

$\mathtt{S} \mathrel{-\!\!\triangleright} \mathtt{G}_2$, $\mathtt{G}_1 \mathrel{-\!\!\triangleright} \mathtt{G}$, and $\mathtt{G}_2 \mathrel{-\!\!\triangleright} \mathtt{G}$, we add a commutativity constraint like:

$$\mathtt{x} \in \mathtt{S}; \mathtt{y}, \mathtt{y}' \in \mathtt{G} : \underline{\mathtt{i}_{\mathtt{G}_1,\mathtt{G}}}(\underline{\mathtt{i}_{\mathtt{S},\mathtt{G}_1}}(\mathtt{x})) = \mathtt{y}, \underline{\mathtt{i}_{\mathtt{G}_2,\mathtt{G}}}(\underline{\mathtt{i}_{\mathtt{S},\mathtt{G}_2}}(\mathtt{x})) = \mathtt{y}' \implies \mathtt{y} = \mathtt{y}'$$

Let us apply this recipe to the class model for file systems in Figure 6. We obtain the signature and constraints `File System` below. Note the seamless integration of the "base type" `String`: On the one hand, it is used as "value-provider" for attributes `name` and `contents`. On the other hand, it is integrated in "graphical" structures as in `contains`. The constraints `a1` and `a2` specify the injectivity of the sub-type embeddings. Constraint `a3` specifies the multiple inheritance of `Directory` and the diamond situation wrt. `Component`. Constraint `a4` specifies that the index, that is used by a `Container` to manage its `Containee`s, is consistent with the naming of the contained `Component`s. Constraint `a5` stems from the `0..1`-multiplicity at the `container`-end of the `contains`-association.

$$\begin{array}{l}
\texttt{File System =}\\
\texttt{sorts:}\\
\ \texttt{C(omponent), C(ontain)er, C(ontain)ee, S(ystem),}\\
\ \texttt{D(irectory), F(ile), L(ink), Str(ing).}\\
\texttt{opns:}\\
\ \underline{\texttt{i}}\texttt{: C'er} \rightarrow \texttt{C, }\underline{\texttt{i}}\texttt{: C'ee} \rightarrow \texttt{C, }\underline{\texttt{i}}\texttt{: S} \rightarrow \texttt{C'er, }\underline{\texttt{i}_1}\texttt{: D} \rightarrow \texttt{C'er,}\\
\ \underline{\texttt{i}_2}\texttt{: D} \rightarrow \texttt{C'ee, }\underline{\texttt{i}}\texttt{: F} \rightarrow \texttt{C'ee, }\underline{\texttt{i}}\texttt{: L} \rightarrow \texttt{C'ee,}\\
\ \underline{\texttt{n(ame)}}\texttt{: C} \rightarrow \texttt{Str, }\underline{\texttt{c(ontents)}}\texttt{: F} \rightarrow \texttt{Str,}\\
\ \texttt{r(erefences): L} \rightarrow \texttt{C'ee, co(ntains): C'er,Str} \rightarrow \texttt{C'ee.}\\
\texttt{axms:}\\
\ \texttt{[a1] }(\texttt{x,x'} \in X\texttt{: }\underline{\texttt{i}}\texttt{(x) = }\underline{\texttt{i}}\texttt{(x')} \Longrightarrow \texttt{x = x'})_{X\in\{\texttt{C'er},\texttt{C'ee},\texttt{S},\texttt{F},\texttt{L}\}}\texttt{,}\\
\ \texttt{[a2] }(\texttt{x,x'} \in \texttt{D: }\underline{\texttt{i}_n}\texttt{(x) = }\underline{\texttt{i}_n}\texttt{(x')} \Longrightarrow \texttt{x = x'})_{n\in\{1,2\}}\texttt{,}\\
\ \texttt{[a3] x} \in \texttt{D;y,z} \in \texttt{C: }\underline{\texttt{i}}\texttt{(}\underline{\texttt{i}_1}\texttt{(x)) = y, }\underline{\texttt{i}}\texttt{(}\underline{\texttt{i}_2}\texttt{(x)) = z} \Longrightarrow \texttt{y = z}\\
\ \texttt{[a4] x} \in \texttt{C'er;s,s'} \in \texttt{Str: }\underline{\texttt{n}}\texttt{(}\underline{\texttt{i}}\texttt{(co(x,s))) = s'} \Longrightarrow \texttt{s = s'}\\
\ \texttt{[a5] x} \in \texttt{C'er;s,s'} \in \texttt{Str: co(x,s) = co(x',s)} \Longrightarrow \texttt{x = x'}
\end{array}$$

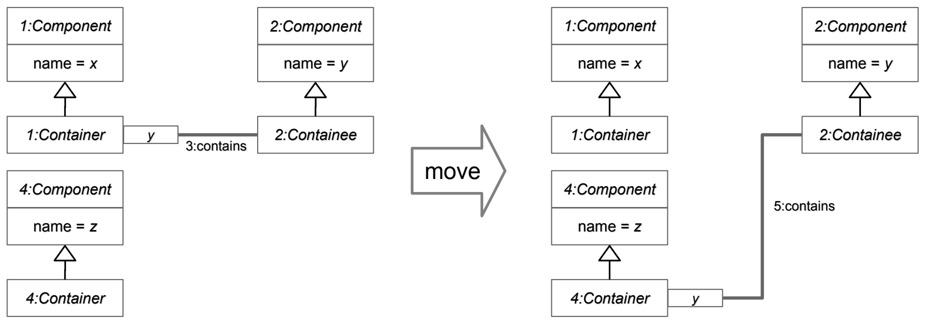

Figure 7. Rewrite Rule: Move Containee

On the basis of the specification `File System`, operations that change the system's structure can be specified by SPO rewrite rules. Figure 7, for example, specifies the operation that moves a `Containee` (2) from `Container` (1) to `Container` (4). Note that this operation fails, if the receiving container already contains a containee named `y`. This is due to the fact that the index "`contains`" is specified as a partial operation that needs to be unique.

The rules in Figure 8 demonstrate the ability of the chosen inheritance model to change the types of objects at "runtime".[23] The rule read from left to right converts a `Link` (2) to a `File` (1) into a file with the same contents as (1). This is possible by exchanging the `Link`-part of object (2) by a `File`-part.[24] The rule read from right to left has the inverse effect, namely it converts a file (2) that happens to have the same contents as another file (1) into a link to the file (1). Note that the context (other `contains`- and `references`-connections) of the manipulated entity on the `Containee`-level is not change, since all `Containee`-parts are preserved.

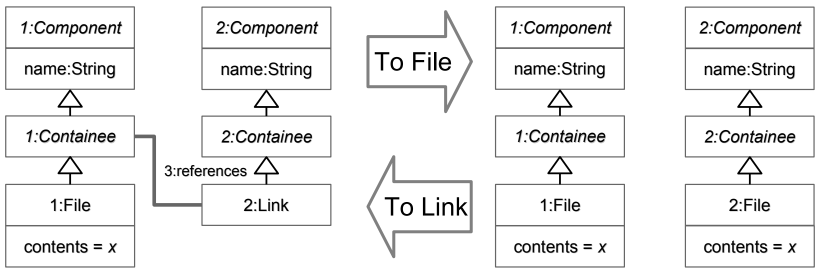

Figure 8. Rewrite Rule: Convert to File / Convert to Link

[23] Runtime means here: In the rewrite process.

[24] Note that the rule first generalises the manipulated object by deleting the `Link`-part. Afterwards it specialises the manipulated object by adding a new `File`-part.

6 Related Work and Future Research

There are only a few articles in the literature that address rewriting of partial algebras, for example [3] and [2] for the double- and single-pushout approach resp. But, both papers stay in the framework of signatures with *unary* operation symbols only and aim at an underlying category of partial morphisms that is co-complete. Aspects of partial algebras occur in all papers that are concerned with relabelling of nodes and edges, for example [10], or that invent mechanisms for exchanging the attribute value without deleting and adding an object, for example [7]. Most of these approaches avoid "real" partial algebras by completing them to total ones by some undefined-values.

Thus, the approach presented in [18] and further developed in this paper is original. It proposes to start with partial algebras in the first place and to require total operations where needed. This is the other way around as most other approaches do: they start with total algebras and add some partiality where needed. Future research will show which approach is more suitable. Another topic for future research is the transfer of SPO-theory to the presented constrained framework. And finally, bigger case studies must be elaborated in order to confirm practical applicability.

References

1. P. Burmeister. *Introduction to Theory and Application of Partial Algebras – Part I*, volume 32 of *Mathematical Research*. Akademie-Verlag, Berlin, 1986.
2. Peter Burmeister, Miquel Monserrat, Francesc Rosselló, and Gabriel Valiente. Algebraic transformation of unary partial algebras II: single-pushout approach. *Theor. Comput. Sci.*, 216(1-2):311–362, 1999.
3. Peter Burmeister, Francesc Rosselló, Joan Torrens, and Gabriel Valiente. Algebraic transformation of unary partial algebras I: double-pushout approach. *Theor. Comput. Sci.*, 184(1-2):145–193, 1997.
4. Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
5. Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Graph Transformations - 6th International Conference, ICGT 2012, Bremen, Germany, September 24-29, 2012. Proceedings*, volume 7562 of *Lecture Notes in Computer Science*. Springer, 2012.
6. Ana Paula Lüdtke Ferreira and Leila Ribeiro. Derivations in object-oriented graph grammars. In Hartmut Ehrig, Gregor Engels, Francesco Parisi-Presicce, and Grzegorz Rozenberg, editors, *ICGT*, volume 3256 of *Lecture Notes in Computer Science*, pages 416–430. Springer, 2004.
7. Ulrike Golas. A general attribution concept for models in M-adhesive transformation systems. In Ehrig et al. [5], pages 187–202.
8. Ulrike Golas, Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Attributed graph transformation with inheritance: Efficient conflict detection and local confluence analysis using abstract critical pairs. *Theor. Comput. Sci.*, 424:46–68, 2012.
9. Esther Guerra and Juan de Lara. Attributed typed triple graph transformation with inheritance in the double pushout approach. Technical Report UC3M-TR-CS-06-01, Technical Report Universidad Carlos III de Madrid, 2006.

10. Annegret Habel and Detlef Plump. M,N-adhesive transformation systems. In Ehrig et al. [5], pages 218–233.
11. Tobias Heindel. Hereditary pushouts reconsidered. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *ICGT*, volume 6372 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2010.
12. Richard Kennaway. Graph rewriting in some categories of partial morphisms. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 1990.
13. Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theor. Comput. Sci.*, 109(1&2):181–224, 1993.
14. Michael Löwe. Algebraic systems. http://ux-02.ha.bib.de/daten/Löwe/Master/TheorieInformationssystem/Algebra20150606.pdf, June 2015.
15. Michael Löwe. Polymorphic sesqui-pushout graph rewriting. In Francesco Parisi-Presicce and Bernhard Westfechtel, editors, *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, volume 9151 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015.
16. Michael Löwe, Harald König, and Christoph Schulz. Polymorphic single-pushout graph transformation. In Stefania Gnesi and Arend Rensink, editors, *FASE*, volume 8411 of *Lecture Notes in Computer Science*, pages 355–369. Springer, 2014.
17. Michael Löwe, Harald König, Christoph Schulz, and Marius Schultchen. Algebraic graph transformations with inheritance and abstraction. *Sci. Comput. Program.*, 107-108:2–18, 2015.
18. Michael Löwe and Marius Tempelmeier. On single-pushout rewriting of partial algebras. *ECEASST*, 73, 2016.
19. Miquel Monserrat, Francesc Rossello, Joan Torrens, and Gabriel Valiente. Single pushout rewriting in categories of spans I: The general setting. Technical Report LSI-97-23-R, Department de Llenguatges i Sistemes Informtics, Universitat Politcnica de Catalunya, 1997.
20. Marius Tempelmeier and Michael Löwe. Single-Pushout Transformation partieller Algebren. Technical Report 2015/1 (in German), FHDW-Hannover, 2015.
21. Wolfgang Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1992.