DRAG REWRITING

NACHUM DERSHOWITZ, JEAN-PIERRE JOUANNAUD, AND FERNANDO OREJAS

Abstract. We present a new and powerful algebraic framework for graph rewriting, based on *drags*, a class of graphs enjoying a novel composition operator. Graphs are embellished with roots and sprouts, which can be wired together to form edges. Drags enjoy a rich algebraic structure with sums and products. Drag rewriting naturally extends graph rewriting, dag rewriting, and term rewriting models.
Keywords: Graph rewriting, drags, composition

Ring-a-ring o' roses,
A pocket full of posies,
A-tishoo! A-tishoo!
We all fall down.

—From: Kate Greenaway,
Mother Goose or
The Old Nursery Rhymes (1881);
illustration from
Harper's Young People (1881)

Date: Draft of July 5, 2023.

Contents

1. Introduction

Rewriting with graphs has a long history in computer science, graphs being used to represent data structures, as well as program structures and even concurrent and distributed computational models. They therefore play a key rôle in program evaluation, transformation, and optimization, and more generally in program analysis; see, for example, [CS18].

As rewriting graphs is very similar to rewriting algebraic terms, the same questions arise: What rewriting relation do we need? Is there an efficient pattern-matching algorithm? How can one determine if a particular rewriting system is confluent and/or terminating?

The above questions have, for these reasons, been addressed by the rewriting community since the mid-seventies in research initiated by Hartmut Ehrig and his collaborators, either in the context of general graphs equipped with pushouts [EPS73], or for particular classes of graphs, specifically those that do not contain cycles [PH94]. In particular, termination and confluence techniques have been elaborated for various generalizations of trees, such as rational trees, directed acyclic graphs, jungles, term-graphs, lambda-terms, and lambda-graphs, as well as for graphs in general. See [Cou90, Cou93] for detailed accounts of these techniques and [HT18] for a survey of implementations of various forms of graph rewriting and of available analysis tools.

Drags, the alternative model considered here, are directed graphs equipped with roots and sprouts that facilitate composition. They can be viewed as networks of processing units that accept a given (finite) number of data as inputs and deliver data as outputs that can be sent over an arbitrary (finite) number of one-way channels. Channels can of course be duplicated, allowing thereby for arbitrary sharing. There is an order among the input channels of a processing unit so as to appropriately discriminate among the inputs. Inputs enter a drag at its sprouts, which are vertices without successors, labeled by variables. Outputs exit the drag at its roots, which are incoming edges with no source. Duplication is modeled by multiplicity of a given root vertex and of a given variable labeling several sprouts. Thus, drags differ from ordinary directed labeled graphs in their distinguishing of roots and sprouts.

The generality of this graph model requires that these one-way channels connect the processing units in an arbitrary way via their respective roots and sprouts. Connecting two drags defines a composition operator so that matching a left-hand side of rule L in a drag D amounts to writing D as the composition of a context graph C with L, and rewriting D with the rule $L \to R$ amounts to replacing L with R in that composition. Composition plays therefore the rôle of both context grafting and substitution in the case of trees, which appear as a simple case of drags. In sharp contrast with term rewriting, however, drag rewriting takes advantage of the underlying graph model: subdrags shared by L and R need not be removed before being (re-) generated. This holds for sprouts as well. On the other hand, distinct sprouts labeled by the same variable must, in the composition, point to *equimorphic* subdrags of the drag to be rewritten, equimorphic drags being identical up to the names of their vertices. This is similar to the double-pushout approach to rewriting (DPO) formalism, which specifies which vertices are to be removed, which are to be (re-) generated, from which variables are absent, and which applies to many different categories of graphs. In our model, this specification is implicit, following the determination of the user that left-hand and right-hand sides of a rule do or do not share specific subgraphs.

One important objective of this work is to make modeling of sharing in graph structures possible so as to enable formal proofs of sharing techniques in programming languages using graphs.

In this work, we completely revamp the preliminary drag model of [DJ19]. In particular, the arrangement of roots in this work is much more useful, variables provide much more flexible sharing, the notion of rewriting rule is much more general, and we end up with a much better algebra. Repeated (nonlinear) variables are used to restrict matches to equimorphic subgraphs. Distinct drags may now share vertices, which is economical when rewriting. The proposed model encompasses term rewriting as a special case, in stark contrast to the prior work. Composition is facilitated by a new notion of step-by-step wiring of connections from sprouts to roots. We develop a pleasant algebra of drags with sum and product. These advances are supported by new, original notions of morphisms for drags, which allow us to precisely relate drag rewriting based on composition with drag rewriting based on DPO, and even to slightly generalize DPO when applied to drags. We observed that seemingly minor changes in the details of the formalism have had far-flung effects and have required significant effort to put all the pieces together in place.

2. The Drag Model

Drags were introduced in [DJ19, DJ18] and developed further in [JO23]. Retaining the moniker, we introduce here a completely new version, which is much better behaved and allows for easier generalization, while at the same time is more simply defined.

Drags are finite ***d***irected ***r***ooted l***a***beled ordered multi-***g***raphs. Some vertices with no outgoing edges are designated *sprouts*. Other vertices are *internal*. We presuppose the following: a set of function symbols Σ, whose elements—equipped with a fixed arity—serve as labels for internal vertices; and a set of nullary variable symbols Ξ, disjoint from Σ, used to label sprouts.

We make considerable use of multisets. A *finite multiset* is a function from a finite base set E to the set $\mathbb{N}$ of natural numbers. Finite multisets are often enumerated as in $\{a, a, b, a, b, c\}$. *Finite sets* are finite multisets all of whose elements occur precisely once as in $\{a, b, c\}$. Given two finite multisets M over E and N over F, a (total) *multi-map* $f : M \to N$ is some (dependent) function $f^+ : (a, m) \mapsto (b, n)$, $0 < m \leq M(a)$, $0 < n \leq N(a)$, so that $f(e)$ will denote the multiset of elements $\{f^+(e, 1), \dots, f^+(e, M(a))\}$. (The second arguments m, n of the function description f^+ serve as indices of the multiple $M(a), N(b)$ occurrences of the first arguments a, b.) A multi-map $f : M \to N$ is *partial* if the associated map f^+ is partial, and *multi-injective* (*multi-equijective*) if the associated map f^+ is injective (bijective, respectively). If M, N are finite sets, then f is a classical injective (bijective) map from M to N. Finally, a map $f : E \to F$ extends to a finite multiset $\{a_i\}_i$ over E as the multi-map returning the finite multiset $\{f(a_i)\}_i$ over F.

To ameliorate notational burden, we use vertical bars $|\_|$ to denote various quantities, such as length of lists, size of expressions, of sets or multisets, and even the arity of function symbols. We use $\varnothing$ for an empty list, set, or multiset, $\cup$ for both set and multiset union (which takes the maximum of multiplicities for multisets),

$\cap$ for set and multiset intersection (minimum for multisets), $\smallsetminus$ for set and multiset difference (natural subtraction for multisets), $\uplus$ for disjoint union (which adds multiplicities), and $\in$ for membership ($a \in M$ iff $M(a) > 0$ for multiset M). We will identify a singleton set or multiset with its single element to avoid unnecessary clutter. So, for example, $a_0 \cup \{a_i\}_{i=1}^{i=n} = \{a_i\}_{i=0}^{n}$.

Definition 1 (Drag). *A* drag *D is a tuple $\langle V, R, L, X, S\rangle$, where*

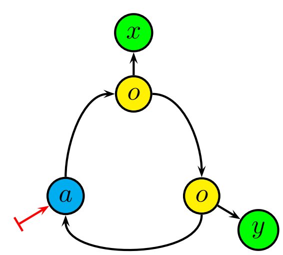
(1) *V is a finite set of* vertices *(vertices have a* name*);*
(2) *$R : V \to \mathbb{N}$ is a finite, possibly empty, multiset of vertices, called* roots*; when $R(v) > 0$, vertex v is* rooted*; when $R(v) = 0$ it's* rootless*; sprouts can be rooted;*
(3) *$S \subseteq V$ is a set of* sprouts*, leaving $I = V \smallsetminus S$ to be the* internal *vertices;*
(4) *$L : V \to \Sigma \cup \Xi$ is the* labeling *function, mapping internal vertices I to labels from vocabulary Σ and sprouts S to labels from vocabulary Ξ (vertices have a* label*);*
(5) *$X : V \to V^*$ is the* successor function*, mapping each vertex $v \in V$ to a list of vertices in V whose length equals the arity of its label, that is, $|X(v)| = |L(v)|$. Sprouts have no successors.*

The pair (R, S) of roots and sprouts is the interface *of drag D.*

Drags are accordingly based on ordered multigraphs with roots.

Definition 2 (Linear; ground; empty; disjoint). *A drag D is* linear *if no two sprouts in $\mathcal{S}(D)$ have the same label,* ground *if it has neither root ($\mathcal{R}(D) = \varnothing$) nor sprout ($\mathcal{S}(D) = \varnothing$), and* empty *(denoted by $\varnothing$) if it has no vertices at all ($\mathcal{V}(D) = \varnothing$). Two drags are* disjoint *if they share neither vertex nor variable.*

Here is an example of a linear drag with three internal vertices (blue and yellow), one (red) root and two (green) sprouts:



Definition 3 (Accessibility). *Drags are directed: If b is the kth vertex in the list $X(a)$ of successors of vertex a of drag $D = \langle V, R, L, X, S\rangle$, then $a \longrightarrow^k b$ is a directed* edge *with* tail *a and* head *b, k being usually omitted. The reflexive-transitive closure X^* of the successor relation X is called* accessibility. *We also write aXb, $a \to b \in X$, or just $a \longrightarrow b$, as well as aX^*b, $a \to b \in X^*$, or $a \longrightarrow^* b$.*

(1) *A vertex v is said to be* accessible from *vertex u, and likewise that u* accesses *v, if uX^*v.*
(2) *Two vertices are* unrelated *if neither is accessible from the other.*
(3) *Accessibility extends to sets as expected, denoting the set of vertices of D that are accessible from any vertex in $W \subseteq V$ by $X^*(W)$.*
(4) *A vertex v is* accessible *(without qualification) if it is accessible from some root, that is if $v \in X^*(r)$ for some rooted vertex r.*

(5) *A* path *of* length n *is a sequence* $u_0, \ldots, u_n$ *of vertices such that* $\forall i \in [0..n-1] : u_i \longrightarrow u_{i+1} \in X$*. The path is* trivial *if* $n = 0$.
(6) *A* cycle *is a (non-trivial) path such that* $s_n = s_0$*. A* loop *is a cycle of length one.*

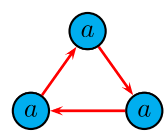
Definition 4 (Connected component)**.** *Given a drag* D *with successor relation* X*, a* connected component *is a subdrag of* D *generated by a set* W *of vertices closed under predecessor, ancestor, and equal labeling of sprouts:* $\forall u \forall v : uXv$ *we have* $u \in W$ *iff* $v \in W$*, and* $\forall s : x \forall t : x$ *we have* $s \in W$ *iff* $t \in W$*.*

Definition 5 (Predecessor; indegree)**.** *We denote by* $pred(v, D)$*, or simply* $pred(v)$*, the number of incoming edges to* v *in drag* D*, and by* $in(v, D)$*, or simply* $in(v)$*, called the* indegree *of* v*, the number of its incoming edges plus the number of times* v *is a root of* D*:* $in(v, D) = pred(v, D) + R(v)$*.*

A vertex is a source *of a drag if it has no predecessor, is a* sink *if it has no successor, and is* isolated *if it has neither predecessor nor successor, hence is both a source and a sink.*

A *tree* is a ground drag all vertices of which have one predecessor, except for a lone vertex with none. A *forest* is a ground drag with no cycle, all vertices of which have zero or one predecessor. A *dag* is a ground drag sans cycles.

Here is a ground drag that is not a dag:



Remark 6. *Two other natural data structures are possible for the roots of drags: lists with repetitions and sets. Sets imply that arbitrarily many output channels can be given access to a given root. Multisets put a precise bound on that number and have slightly better algebraic behavior compared to lists with repetitions, which were used in* [DJ19]*. Completing the set of natural numbers with an infinite value allows one to easily encode set-like behavior by a multiset, another reason for our choice here.*

Remark 7. *Another important difference vis-à-vis* [DJ19] *is that we now also consider drags with inaccessible vertices, such as ground drags* all *of whose vertices are inaccessible.*

Notations:

- When convenient, a drag $D = \langle V, R, L, X, S \rangle$ will be denoted
$$\langle \mathcal{V}(D), \mathcal{R}(D), \mathcal{S}(D), \mathcal{L}(D), X(D) \rangle$$
with $\mathcal{I}nt(D)$ being its internal vertices, $\mathcal{A}cc(D)$ its set of accessible vertices, $\mathcal{V}ar(D)$ the set of variables labeling its sprouts, and $\mathcal{D}om(R)$ the domain of the partial function R.
- We write $r^{[n]} \in R$ to indicate that there are n copies of the rooted vertex r in the multiset R, that is, $R(r) = n$, but also $r \in R$, considering R as the set of rooted vertices. A root may also be seen as an edge without designated tail; so we will sometimes use the notation $\longrightarrow r$ or $\longmapsto r$ to indicate that vertex r is rooted.

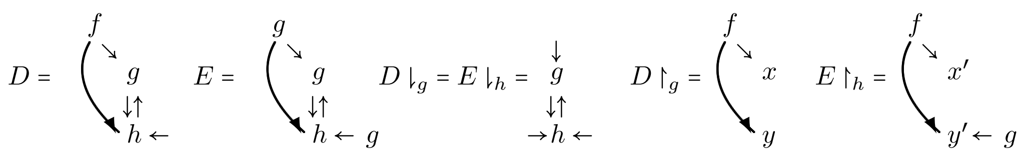


FIGURE 1. Two drags with the same subdrag but different context drags.

- We write $u : f$ when the vertex u (possibly a sprout) has label f (possibly a variable). The labeling function extends to lists, sets, and multisets of vertices as expected.
- In examples, we will often name vertices by their label, when the intention is clear. In case of ambiguity, we will index the label by a positive number, so that $f(x, x)$ has vertices f, x_1, and x_2. Combining these notations, $f^{[2]}(x, x^{[1]})$ has now two roots at vertex f and one root at vertex x_2. In that case, we will alternatively write $f^{[2]}(x_1, x_2^{[1]})$.
- In pictures, roots will be illustrated by incoming arrows, possibly indexed by a natural number indicating their multiplicity.

3. Contexts and Subdrags

Definition 8 (Subdrag; context)**.** *Let D be the drag $\langle V, R, L, X, S\rangle$, and let $W \subseteq V$. We define the following notions:*

(1) *The* restriction $D|_W$ *of drag D to vertices W is the drag*

$$D' = \langle W \cup S', R', L', X', (W \cap S) \cup S'\rangle$$

where

(a) $S' = \{s_v : v \in V \setminus W, \exists w \in W : wXv\}$ *are new sprouts, the s_v being new vertices;*

(b) $R'(w) = R(w) + \Sigma_{v \longrightarrow^k w \in X, v \in V \setminus W} k$*, for each $w \in W$ (hence $in(w, D') = in(w, D)$);*

(c) *L' coincides with L on W, while $L'(s_v) = x_v$ for each $s_v \in S'$, where x_v is a fresh variable;*

(d) *X' coincides with X on W, and for each $s_v \in S'$, $u \longrightarrow^k s_v \in X'$ iff $u \longrightarrow^k v \in X$.*

(2) *The* subdrag $D \downharpoonleft_W$ *of D* generated *by W is the restriction of D to the set of all vertices accessible from W. That is, $D \downharpoonleft_W = D|_{X^*(W)}$. The subdrag is* void *when $W = \varnothing$,* trivial *when $X^*(W) = V$ (as for $D \downharpoonleft_R$ because all vertices of D are accessible from R), and* strict *when not trivial.*

(3) *The* context $D \upharpoonleft_W$ *of W in D is the restriction of D to the set of vertices that are inaccessible from W, viz. $D|_{V \setminus X^*(W)}$.*

Let D a drag reduced to a single internal vertex v and edge $v \longrightarrow v$. Then, the restriction of D to v is D itself. Examples of drags, subdrags, and context drags are shown in Figure 1.

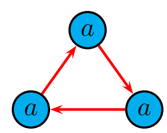
Subdrags need no new sprouts since their vertices are closed under succession, but context drags do. On the other hand, a (nontrivial, non-void) subdrag always has new roots. In particular, a nontrivial subdrag of a term at some position in the term has a root at its head, while the subterm has none. These new roots play an important rôle in the reconstruction of D from $D|_W$ and $D\restriction_W$.

Lemma 9. *The strict subdrag relation is a well-founded order.*

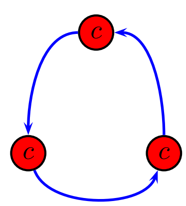
Proof. Because strict subdrags have fewer vertices. ☐

4. An Example

To motivate the development of drag rewriting, consider an example. The goal is to take a ring of blue vertices (a ground drag) like this:



and create instead a ring consisting of red vertices, in the same quantity as the blue but going in the opposite direction, like this:

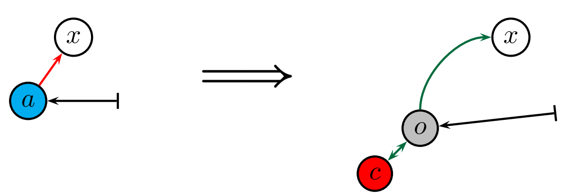


To make what's happening clearer, we will be using red for original edges, blue for new, and green for temporary ones.

We seek a rewriting algorithm that can apply at the same time—in parallel—to many vertices along the ring.

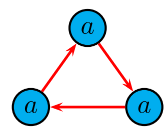
The left-hand and right-hand sides are drags; see [DJ19]. Left roots map to right roots, and the two share variables.

There are two rules. The first creates the red vertex and introduces a gray vertex to keep track of connections.

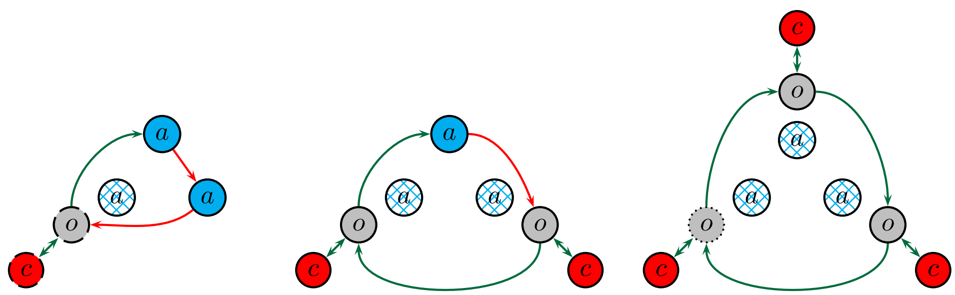


The two new vertices, red c and gray o, are added with edges connecting them in both directions. The edge from a to whatever x may be is replaced by an edge from o. Instead of the root a on the left, it is o that becomes the new root. Vertex a becomes detached as its incident edges are removed by the rule.

Applying this rule thrice to

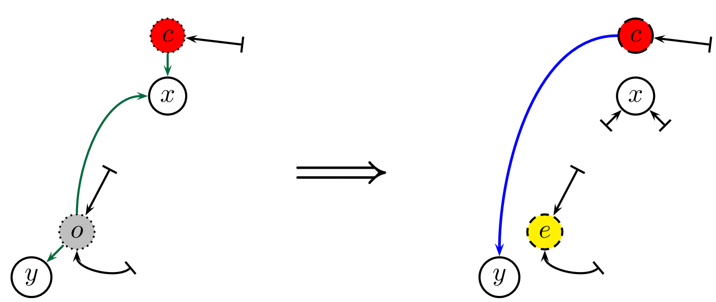


starting at the lower left and proceeding counterclockwise, we get the following sequence of drags:



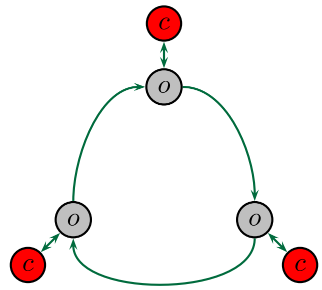
The orphaned a vertices are left shaded.

The next rule connects the added red vertices in the opposite direction:

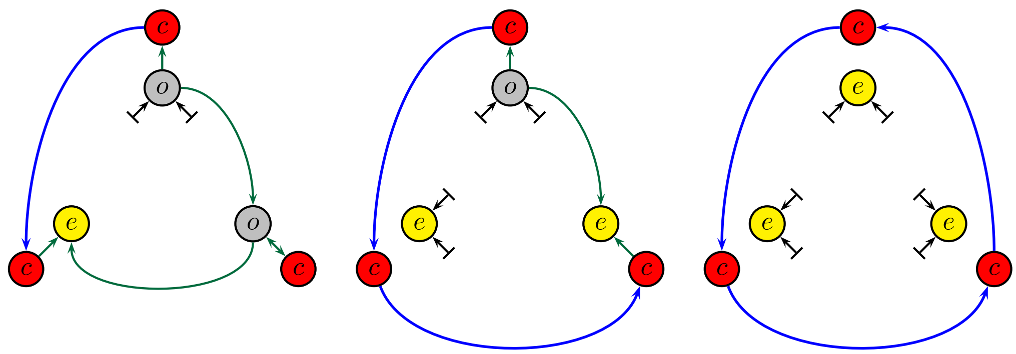


The red c vertex on the left is dotted and the one on the right is dashed to indicate that they are actually distinct vertices. The edge from it is redirected (in blue) to the other vertex pointed to by the gray o that points to the original head of the edge from c. A new, yellow double-rooted vertex e is created to replace the deleted, double-rooted o, and serves to preserve incoming edges. The two edges to the vertex signified by variable x have been cut, so are now just roots.

Applying this rule to

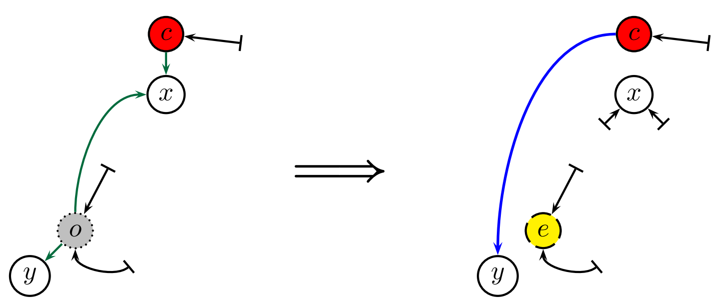


three times, we get



This rule cannot be applied before a red vertex is created by the previous rule. Later on, we will consider garbage collection for vertices like e that have served out their purpose. Actually, the old c vertex also sticks around and can be recycled.

An alternative is to allow the left-hand and right-hand drags to share internal vertices—not just sprouts, but to have separate sets of edges for the two sides. The two rules are the same as before, except that now the red vertex c in the second rule is shared by both sides:



The red c vertex on the two sides has a solid border to make it clear that it is the same vertex on the two sides. The edge from c to x is redirected to y, as before. As in the previous version, the gray o vertex is only in the left drag, while the yellow e is only in the right one. The application of this rule would look the same as the above, except that the red vertices are preserved by the rule.

Later, in Section 13.2, we will mention another possible style of rules, with sharing of subdrags between left- and right-hand sides.

5. Drag Morphisms

A vertex of a drag has both a name (an element in V) and a label, taken from Σ or Ξ. In particular, the sprouts of a drag are vertices labeled by variables from Ξ. The vertices of ordinary terms, on the other hand, are usually left nameless, with their positions (in a Dewey-decimal-like notation) standing in for names. In the term tradition, sprouts *are* variables: the difference between a sprout and its label does not matter because the term framework does not distinguish between two terms that correspond to distinct isomorphic graphs. Drags being graphs, two drags may be identical or they may be isomorphic as graphs. This distinction becomes crucial when it comes to sharing, which is why it is not relevant for terms, where there is no sharing. Of course, terms can be seen as a particular kind of drag, but, as just stressed, it is important to understand that term equality for trees corresponds to isomorphism of drags, not identity.

To define precisely the kinds of equalities on drags in which we are interested, notions of drag morphism, drag monomorphism, and drag isomorphism are required. These must of course reduce to the corresponding notions on graphs for ground drags. The possibility that drags share vertices will require special care. Another important matter is categoricity.

As usual, morphisms will be maps from the set of edges of the input drag to the set of edges of the target drag that are the identity on shared vertices. What differs from the usual graph morphisms is that we need to take care of variables, regarding which there will be three problems. First, internal vertices need be mapped to internal vertices, as is already the case with terms. Second, the drag may be nonlinear, two or more sprouts being labeled the same. Such sprouts cannot be mapped independently of each other. Third, two vertices of the input drag may be mapped to the same vertex of the target drag in case the mapping is not injective. This is a standard situation for morphisms, but this has to happen for monomorphisms as well, if only for two sprouts sharing the same label. But monomorphisms may also have to map a sprout and an internal vertex of the input drag to the same vertex of the target drag: Monomorphisms will be injective on internal vertices only. We address the two latter questions in turn.

In the sequel, we consider two arbitrary drags D, D' that possibly share vertices. We make the implicit assumption throughout the paper that sharing propagates to the successors of shared vertices. In other words, if a vertex is shared, then all its reachable vertices are shared between the drags, too.

Definition 10 (Premorphism)**.** *Given two drags* $D = \langle V, R, L, X, S\rangle$ *and* $D' = \langle V', R', L', X', S'\rangle$*,* I *and* I' *their respective sets of internal vertices, a* premorphism *from* D *to* D' *is a map* $o : V \to V'$*, called a* vertex map*, which (1) restricts to mappings from internal vertices* I *to* I' *(denoting this restriction by* o_I*) and preserves their labels, (2) restricts to mappings from isolated sprouts of* D *to isolated sprouts of* D'*, (3) is the identity on shared vertices, and (4) has the property on edges that* $u' \longrightarrow^i v' \in X'$ *iff* $u \longrightarrow^i v \in X$ *for some vertices* u, v *such that* $u' = o(u)$ *and* $v' = o(v)$*. We define the* edge map *of the premorphism as* $o_X(u \longrightarrow^i v) = o(u) \longrightarrow^i o(v)$*.*

In case D and D' are ground drags, premorphisms are just ordinary graph morphisms.

Definition 11 (Equimorphism)**.** *Given two drags* $D = \langle V, R, L, X, S\rangle$ *and* $D' = \langle V', R', L', X', S'\rangle$*, an* equimorphism *is a premorphism* o *that is bijective, preserves labels at all vertices [*$L'(o(u)) = L(u)$ *for all* $u \in V$*], preserves all edges [*$o(u) \longrightarrow^i o(v) \in X'$ *iff* $u \longrightarrow^i v \in X$*], and preserves roots at all vertices [*$R'(o(u)) = R(u)$ *for all* $u \in V$*].*

Given a non-injective premorphism between drags D and D' that possibly share vertices, a difficulty is to make explicit the relationship between the roots of D and those of D'. To this end, we need to categorize edges into several subsets.

Definition 12 (Outside/entering/created/mapped edges)**.** *Given drags* $D = \langle V, R, L, X, S\rangle$ *and* $D' = \langle V', R', L', X', S'\rangle$*, with* I *and* I' *their respective internal vertices, a premorphism* $o : V \to V'$*, an internal vertex* $u' \in I'$*, and an edge* $u' \longrightarrow^i v' \in X'$*, we use the following terminology:*

(1) *Edge* $u' \longrightarrow^i v'$ *is an* outside edge *if* u' *is not the image by* o *of an internal vertex of* D*, and either* v' *is not an image of an internal vertex of* D *or else* $v' \in S' \setminus S$*.*
(2) *Edge* $u' \longrightarrow^i v'$ *is an* entering edge *of* D' *at* $v' = o(v)$ *if* u' *is not the image by* o *of a vertex of* D*, and* v' *is the image of an internal vertex of* D *or else* $v' \in S' \cap S$ *(hence* $u' \neq v'$ *in both cases). We denote by*
   - $\#ee(v')$ *the number of entering edges of* D' *at* v'*.*
(3) *Edge* $u' \longrightarrow^i v'$ *is* created *by the* creating edge $u \longrightarrow^i s \in X$ *if* u' *is the image by* o *of some internal vertex* u *of* D *and* s *is a sprout such that* $o(s) = v' \in V' \setminus V$*. We denote by*
   - $Sce(v')$ *the set of all creating edges of* D *at* v'*, and by*
   - $\#ce(v')$ *its size.*
(4) *Edge* $u' \longrightarrow^i v'$ *is* mapped from inside edge $u \longrightarrow^i v$ *such that* $o_X(u \longrightarrow^i v) = u' \longrightarrow^i v'$ *(hence* u' *is the image by* o *of internal vertex* u*) if either* $v' \in I'$ *or else* $v' \in S \cap S'$*. We denote by*
   - $\#me(v')$ *the number of inside edges mapped to an edge with head* v'*.*

We additionally define the set of contributing *vertices of* D *at arbitrary vertex* v' *of* D' *as*

- $Scv(v') = \begin{cases} \{v \in I : o(v) = v'\} & \text{if } v' \in V' \setminus (S \cap S') \\ \{s \in S : o(s) = s = v'\} & \text{if } v' \in S \cap S', \end{cases}$

and denote the total number of its (contributed) roots by

- $\#cr(v')$.

Note that the set $Scv(v')$ of contributing vertices of D at v' is empty if v' is a sprout not shared by both drags, since the inverse image of a sprout is never an internal vertex, and it is that sprout if v' is shared. If v' is an internal vertex, all internal vertices v of D such that $v' = o(v)$ are contributing vertices of D at v'.

In case o is not injective, there may be several edges $u \longrightarrow^i s$ for the same created edge $o(u) \longrightarrow^i v'$; hence created edges form a multiset, while creating edges form a set, of course. Likewise, several edges of D forming a set may be mapped to the same inside edge of D', forming a multiset. Furthermore, the same edge of D' can be both a created edge and an inside edge obtained, possibly several times, from

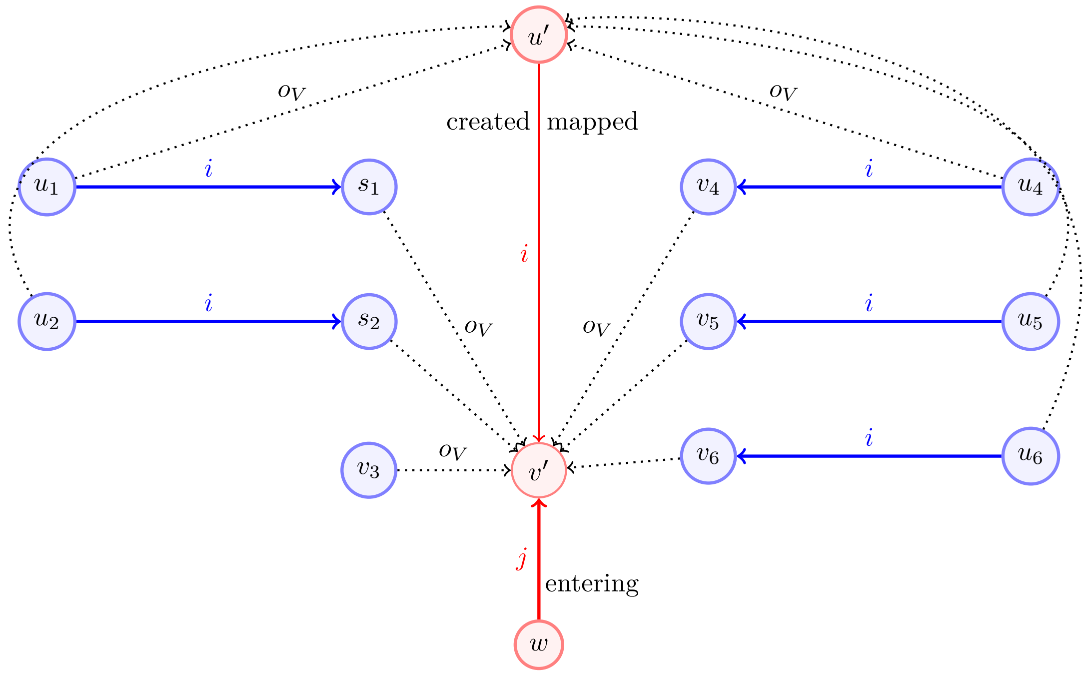


FIGURE 2. Creating (left), entering (below) and mapped (right) edges. Vertices and edges are blue for D and red for D'.

different edges of D. Entering edges always form a set. On the D side, an edge is either mapped at or creates an edge of D'. The situation is illustrated in Figure 2, vertex v' being the head of an edge $u' \longrightarrow^i v'$ of D' created twice and mapped thrice, and is the head of an entering edge $w \longrightarrow^j v'$ ($j = i$ is of course possible), which is unique given w and j.

If o is injective, there cannot be, for a given edge $u' \longrightarrow^i v' \in X'$, two different creating edges $u_1 \longrightarrow^i s_1$ and $u_2 \longrightarrow^i s_2$, nor two different mapped edges $u_1 \longrightarrow^i v_1$ and $u_2 \longrightarrow^i v_2$, nor a creating edge $u_1 \longrightarrow^i s_1$ and a mapped edge $u_2 \longrightarrow^i v_2$. In that case, the inverse image u of u' is a unique internal vertex, implying that the pair (u, i) is unique as well.

Finally, "exiting" edges do not exist: Imagine $u' \longrightarrow^i v'$ to be an edge of D' such that $u' = o(u)$, for $u \in I$, while v' is not the image of any $v \in I$. Then, there must be some edge $u \longrightarrow^i w$ in D by the definition of premorphisms, implying that $v = o(w)$, a contradiction.

Definition 13 (Preserving)**.** *A premorphism o between drags D and D'*

- preserves *equimorphic subdrags if it maps equimorphic subdrags of the source drag to equimorphic subdrags of the target drag;*
- preserves *roots at vertex v' of D' such that $Sce(v') \neq \varnothing$ if $R(v') = \#cr(v') - \#ce(v') - \#ee(v')$.*

The idea behind root preservation is that an edge is created in D' from both a creating edge of D and a root at some contributing vertex of D. Likewise, a root at a contributing vertex is also needed to form the entering edges of D'. Vertices of D' whose set of contributing vertices is empty have no antecedent in D by o, apart

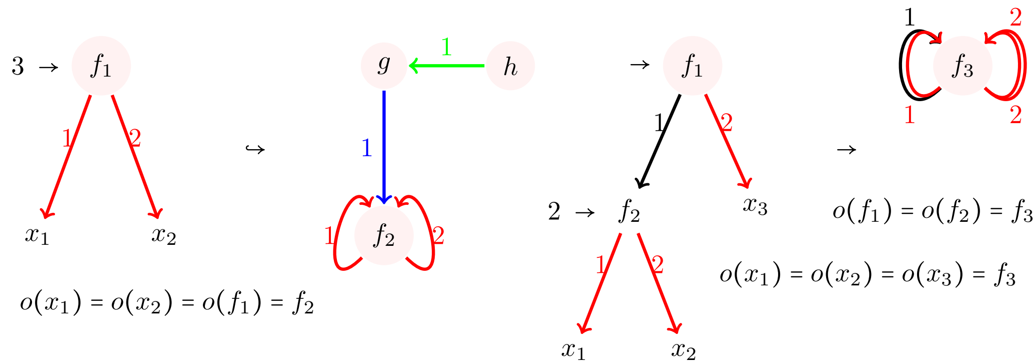


FIGURE 3. A monomorphism (↪) on the left and a morphism (→) on the right. Edges: black for mapped, red for created, blue for entering, green for outside. Incoming arrows at some vertex annotated with numbers stand for multiple roots.

from non-shared sprouts; their roots don't serve to establish new edges. This will become apparent when studying matching in Section 12, which makes heavy use of edge categorization and root preservation.

Definition 14 (Morphism)**.** *A* morphism *from drag D to drag D' is a premorphism that preserves roots at all contributing vertices, and preserves equimorphic subdrags.*

Definition 15 (Monomorphism, isomorphism)**.** *A* monomorphism *is a morphism whose vertex map restricts injectively to internal vertices and isolated sprouts. An* injection *is a monomorphism whose vertex map is the identity on internal vertices and isolated sprouts. An* isomorphism *is a monomorphism whose vertex map is bijective.*

Since an edge $o(u) \longrightarrow^i o(v)$ is characterized by the pair $(o(u), i)$, which is unique when o is injective, the edge map o_X of a monomorphism o is injective on all edges. That's why we do not need to state it, even though o is not injective on all vertices. This would not be the case were an edge simply a pair $u \longrightarrow v$ instead of a triple $u \longrightarrow^i v$.

Just like monomorphisms on terms, monomorphisms on drags relate to matching. Thinking in terms of matching is useful for understanding the subtleties of our categorization of edges, which governs the notion of root preservation used for defining morphisms. The following commented examples illustrate this categorization of edges.

Example 16 (Morphism)**.** *Consider drags $D = f^{[1]}(f^{[2]}(x, x), x)$, with vertices f_1 (above) and f_2 (below) labeled by the binary symbol f and three other vertices x_1, x_2, and x_3 (in depth-first order), and $D' = f(\textsc{self}, \textsc{self})$, a drag with a single vertex f_3 labeled f, two edges $f_3 \longrightarrow^1 f_3$ and $f_3 \longrightarrow^2 f_3$, and no root. Observe that the mapping $o : D \to D'$ such that $o(f_1) = o(f_2) = o(x_1) = o(x_2) = o(x_3) = f_3$ is a premorphism. This example is represented on the right of Figure 3.*

Edges $f_3 \longrightarrow^1 f_3$, $f_3 \longrightarrow^2 f_3$ are created edges of D', the first being created once (from edge $f_2 \longrightarrow^1 x_1$) and the second twice (from edges $f_1 \longrightarrow^2 x_3$ and $f_2 \longrightarrow^2 x_2$).

But the first is also obtained twice, as both created and mapped by o_X *(from the edge* $f_1 \longrightarrow^1 f_2$*).*

We show that root preservation holds at f_3*. The number of roots at* f_3 *is* 0*. The number of mapped edges is* 1 *(they don't count for root preservation). The number of entering edges is* 0*. The number of creating edges is* 3*. And the number of contributed roots at* f_3 *is* 3*.*

Since root preservation is satisfied (0=3-3-0), o is a morphism. □

Example 17 (Monomorphism)**.** *The same Figure 3 shows on its left a morphism that maps the internal vertex* f_1 *to vertex* f_2*, and both sprouts* x_1 *and* x_2 *to the internal vertex* f_2*. The right drag has one outside edge issuing from vertex* h*, one entering edge issuing from vertex* g*, and two created edges* $f_2 \longrightarrow^1 f_2$ *and* $f_2 \longrightarrow^2 f_2$ *originating from two creating edges* $f_1 \longrightarrow^1 x_1$ *and* $f_1 \longrightarrow^2 x_2$ *of the left drag. We are left with checking root preservation for* f_2*, which has no root, and a single contributing vertex,* f_1*, which has three roots, that is:* $0 = 3 - 2 - 1$*, which is therefore satisfied. We have a monomorphism.*

Example 18. *Figure 4 (left) is an example of a monomorphism [*$o(f_1) = f_2$*,* $o(a_1) = a_2$*,* $o(x) = a_2$*] with all kinds of edges. Note that the left vertex labeled a has lost its root, used by the creating edge* $f_1 \longrightarrow^1 x$ *to create the edge* $f_2 \longrightarrow^1 a_2$*. In this example, the set of contributing vertices is reduced to the vertex* a_1*, the inverse image of* a_2*.*

On the right, all edges are creating (for the left drag) or created (for the right drag). Note that a has two roots, but it could have fewer or more. The reason is that a has no inverse image that is internal in the left drag: its set of contributing vertices is empty. Thinking in terms of matching, vertex a belongs to the matching context and may come along with any number of roots it has in the context.

In Figure 5, $o(f_1) = f_2$*,* $o(h_1) = h_3$*,* $o(h_2) = h_4$*, and* $o(x_1) = o(x_2) = o(z) = y$*. In this example, sprout* y *is shared by both drags, hence does not belong to the context:* y *is its own set of contributing vertices, and its number of roots is then determined by the left drag. Vertex* y *has four roots in left drag, but there are three creating edges which require those three roots, which have therefore disappeared in the right drag, only one is left.*

Figure 6 shows a variation of the example of Figure 5, where sprout y*, whose number of roots is now unimportant, is mapped to a non-shared sprout* t *(it could be an internal vertex as well), and a vertex* g *has been added so as to have an outside edge* $g \longrightarrow^1 t$ *(in green), which shows clearly that both vertices* g *and* t *originate from the context, hence may have any number of roots one likes.*

Example 19 (Isomorphism)**.** *We now show that the two drags* $D = f(x, y^{[1]})$ *and* $D' = f(x, z^{[1]})$*, sharing the sprout labeled* x*, where* y *and* z *are different, are isomorphic. Using the map* $o : D \to D'$ *such that* $o(f_1) = f_2$*,* $o(x) = x$ *and* $o(y) = z$*, o being the identity for the shared vertex* x*, implying that* $o_X(f_1 \longrightarrow^1 x) = f_2 \longrightarrow^1 x$ *and* $o_X(f_1 \longrightarrow^2 x) = f_2 \longrightarrow^2 z$*, we get* $o(D) = D'$*. Note that* D *has no created or entering edge, hence the root* $\longrightarrow y$ *in* D *is not utilized for compensation, it can be identified to the root* $\longrightarrow z$ *of* D'*. The map o is clearly a monomorphism that is bijective and preserves roots and edges, hence is an isomorphism. Were the vertex* z *in* v *replaced by a vertex* w *labeled* y*, both drags would be equimorphic, regardless of whether* w *is shared with the vertex* y *in* u*.* □

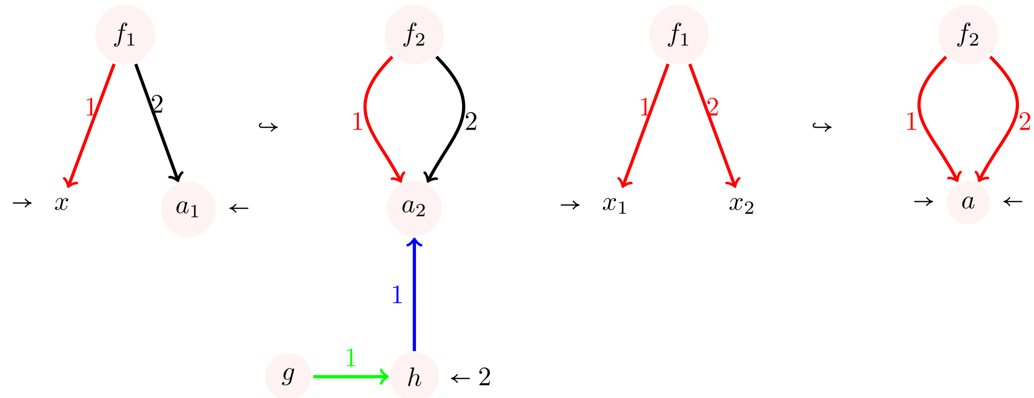
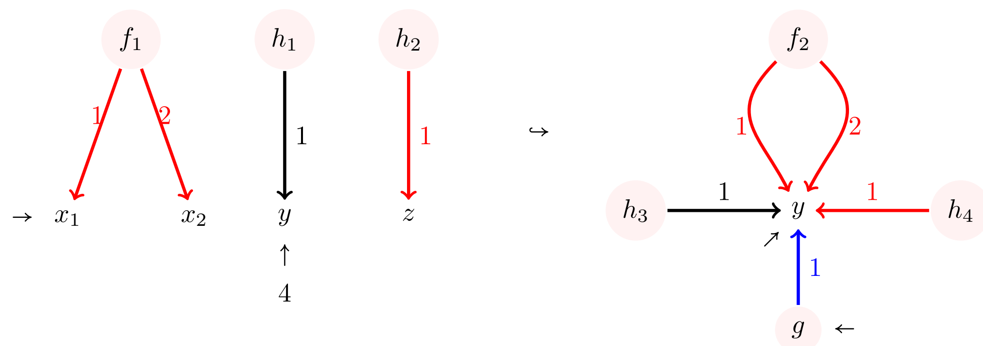


FIGURE 4. Additional examples of monomorphisms.



$$o(f_1) = f_2, o(h_1) = h_3, o(h_2) = h_4, o(x_1) = o(x_2) = o(z) = o(y) = y$$

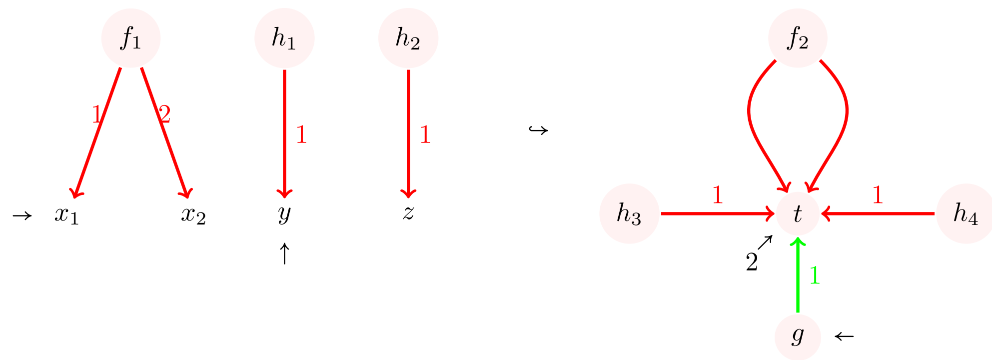
FIGURE 5. A monomorphism with shared sprout y.

Monomorphisms enjoy an additional key preservation property equivalent to root preservation, which we may use without saying in the sequel:

Lemma 20. *Let D and D' be two drags with respective vertices V and V', and o a morphism from D to D' injective on internal vertices and preserving equimorphic subdrags. Then, o is root preserving (hence is a monomorphism) iff it preserves indegrees, that is,*

$$\forall v' \in V' \text{ such that } v \in Scv(v') : in(v, D) = in(v', D').$$

Proof. Note first that $v \in Scv(v')$ is unique if it is internal since o is injective on internal vertices, and unique as well if $v = o(v)$ is a shared sprout, the only two cases for which a contributing vertex exists.



$$o(f_1) = f_2, o(h_1) = h_3, o(h_2) = h_4, o(x_1) = o(x_2) = o(y) = o(z) = t$$

FIGURE 6. A monomorphism with a green outside edge.

Assuming first that o is root preserving, we have

$$\begin{aligned} in(v', D') &= pred(v', D') + R'(v') \\ &= (\#me(v) + \#ce(v) + \#ee(v)) + (\#cr(v) - \#ce(v) - \#ee(v)) \\ &= \#cr(v) + \#me(v) \\ &= R(v) + pred(v) = in(v, D). \end{aligned}$$

Assuming conversely that $in(v', D') = in(v, D)$, the same sequence of equalities shows root preservation for the pair (v, v'). □

We write $o : D \rightarrow D'$ if o is a premorphism or morphism, $o : D \hookrightarrow D'$ or $D \subseteq_o D'$ if o is a monomorphism, $D \simeq_o D'$ if o is an isomorphism, $D \equiv_o D'$ if o is an equimorphism, and $D = D'$ if o is identity. The subscript o will often be omitted. Monomorphisms may be abbreviated "monos".

The action of morphisms on sprouts or roots is a specificity of the notion of (nonground) drags. A root being seen as a potential edge, it must be mapped to either a root if it is unutilized, or to an edge if it is used to build an edge. Sprouts of the same labeling being equimorphic, they must be mapped by morphisms to equimorphic subdrags, possibly sprouts of the same labeling. This condition on sprouts is vital to ensure that morphisms compose.

Note the difference between a monomorphism and an injection: both preserve labels of internal vertices, but injections also preserve their names, while monomorphisms need not. Like monomorphisms, injections may have entering as well as created edges.

Being a bijection at all vertices, an isomorphism must restrict to sprouts; hence it maps each sprout of D labeled by some variable x to a sprout of D' labeled by some variable y possibly different from x. It follows that isomorphisms have no entering edges, and no created edges either. They must therefore preserve root multiplicity at all vertices; hence the isomorphism restricts to roots as well. Isomorphisms are therefore morphisms whose restrictions to internal vertices, sprouts, and roots are all bijections. Like monomorphisms, isomorphisms do not preserve names of vertices. For that reason, a drag isomorphism does not render the drags equal.

Equimorphic drags are copies of one another, possibly sharing subdrags or being even identical. Therefore, equimorphisms preserve equimorphic subdrags, hence are morphisms, and, being bijective, they are isomorphisms that preserve the labeling of sprouts.

We have already pointed out the strong preservation condition of equimorphic drags by morphisms: The weaker condition that identically labeled sprouts be related by isomorphism, as suggested in [DJ19], would not ensure the following key properties:

Lemma 21. *Morphisms, monomorphisms, injections, isomorphisms, and equimorphisms are closed under composition.*

Proof. Consider $o : D \to D'$ and $o' : D' \to D''$, and let $o'' = o' \circ o : D \to D''$ be their composition.

Premorphisms compose because restrictions to internal vertices and isolated sprouts compose, and o'' is the identity on shared vertices by the definition of composition.

Equimorphisms compose since bijections do, and preservation properties are transitive.

Morphisms compose: Being an equality property, root preservation is transitive. Since equimorphisms compose, preservation of equimorphic subdrags is transitive as well.

Monomorphisms compose: Injectivity on internal vertices and isolated sprouts is preserved by composition.

Being monomorphisms whose vertex map is the identity, injections compose.

Being monomorphisms whose vertex map is bijective, isomorphisms compose. □

Classically, graphs and their morphisms form a category but have no roots at their internal vertices nor variables at their leaves that can be redirected to the roots. Their presence in drags and the associated preservation properties that morphisms must satisfy make nontrivial two properties that are usually obtained naturally.

Drag morphisms are indeed defined on equivalence classes of drags modulo isomorphisms because, given any input drag, an isomorphism preserves its roots at all vertices and maps its equimorphic subdrags to equimorphic subdrags of the output drag.

Monomorphisms on graphs are just injective morphisms between their sets of vertices and between their sets of edges. Here, monomorphisms are more complex maps, being injective on internal vertices only, as are substitutions on terms, which actually implies that o_X is injective on all edges thanks to the natural number component of an edge. These properties imply that our monomorphisms are indeed monomorphisms in the categorical sense, that is, are the left-cancellative morphisms:

Lemma 22 (Categoricity)**.** *Let D', D'' be drags. A morphism κ from D' to D'' is a monomorphism iff for all drags D and morphisms $o, o' : D \to D'$ such that*

$$\kappa \circ o = \kappa \circ o' \qquad (*)$$

it is the case that $o = o'$.

Proof. Claim: Let I and I' be the sets of internal vertices and isolated sprouts of D and D', respectively. Property (*) implies $\kappa_{I'} \circ o_I = \kappa_{I'} \circ o'_I$. Since categoricity reduces to injectivity for functions on sets, it follows that $o_I = o'_I$ iff $\kappa_{I'}$ is injective.

Assume now that κ is a monomorphism, hence that $\kappa_{I'}$ is injective and $o_I = o'_I$ by the above Claim. Therefore, $o_X = o'_X$ by definition of an edge map, which implies in turn that $o_V = o'_V$ for non-isolated sprouts. Altogether, we get $o = o'$.

Conversely, assume that $o = o'$ for all morphisms κ satisfying (*). By the same token as previously, the restriction $\kappa_{I'}$ of the morphism κ is therefore injective, hence κ is a monomorphism. □

The proof shows that injectivity of monomorphisms for internal vertices and isolated sprouts ensures categoricity. The definition of morphisms for sprouts is therefore unimportant, provided transitivity is satisfied. The precise definition we gave has therefore a single objective: to ensure that matching behaves as desired. We will show in Section 12 that this is indeed the case.

We can now conclude:

Theorem 23. *Drags equipped with their morphisms, monomorphisms, and isomorphisms form a category, of which the category of terms is a particular case.*

Notations: We end this section by introducing notations for the various kinds of morphisms and related equalities that are relevant for drags.

We write $o : D \to D'$ for premorphism or morphism o; $o : D \hookrightarrow D'$ ~~or $D \subseteq_o D'$~~ if o is a monomorphism; $D \simeq_o D'$ when o is an isomorphism; $D \equiv_o D'$ if o is an equimorphism; and $D = D'$ when o is identity. The subscript o will often be omitted. Monomorphisms may be abbreviated “monos”.

We may need more precise notations in case of an isomorphism o, writing then $D \simeq_o^\sigma D'$, where σ is a bijection between the variables of corresponding sprouts. If D and D' are equimorphic, then σ is the identity, so $D \simeq_o^{id} D'$ and $D \equiv_o D'$ are indeed the same. The notation $D = D'$ (not to be confused with definitional equality) is reserved for the case where D and D' are the very same drag.

Two drags D and D' are *disjoint* if they share no vertices nor variables. *Renaming apart* two drags D and D' amounts to renaming bijectively the shared vertices and variables of D' so that D'' is isomorphic to D' while D and D'' become disjoint.

6. Drag Operations

There are two main operations on drags, *sum* and *product*, that are reminiscent of similar operations used in the literature, although in restricted settings compared to here. The third operation, *wiring*, is in some sense more fundamental than product, which amounts to a particular case of wiring a sum.

6.1. **Sum.**

Our first operation on drags is a very simple, familiar one when both drags are disjoint: It consists of placing two drags side by side to form a new drag.

Adding together drags that share vertices and edges requires an assumption:

Definition 24 (Compatible drags)**.** *Two drags $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ are* compatible *if (i) $V \cap V'$ is closed under X and X', (ii) L and L' as well as R and R' coincide on $V \cap V'$, (iii) $s : x \in S$ and $t : x \in S'$ for the same variable x implies $s, t \in V \cap V'$, and (iv) D and D' have the same indegree at each shared vertex v, at least equal to the total number of shared (counting for one each) and non-shared edges heading at v.*

In words, compatible drags that share a vertex must also share the whole subdrag generated by that vertex. Similarly, sharing a variable implies sharing the corresponding sprouts. Also, they must have enough roots at their shared vertices so as to have the same indegree after replacing the roots of each graph at a shared vertex by the incoming edges of the other graph at that vertex.

Definition 25 (Sum)**.** *The* parallel composition *or* sum $D \oplus D'$ *of two compatible drags* $D = \langle V, R, L, X, S\rangle$ *and* $D' = \langle V', R', L', X', S'\rangle$ *is the drag* $\langle V \cup V', R'', L \cup L', X \cup X', S \cup S'\rangle$*, where,* $\forall x \in V \cup V' : R''(v) = R(v) - |\{u \longrightarrow^i v \in X' \smallsetminus X\}| = R'(v) - |\{u \longrightarrow^i v \in X \smallsetminus X'\}|$.

Note that the equality statement when defining the number of roots at all vertices in the union of two drags follows from the definition of compatibility. Note also that $R''(v) = R(v)$ if $v \in V \smallsetminus V'$ and $R''(v) = R'(v)$ if $v \in V' \smallsetminus V$, implying that the union of disjoint drags is their juxtaposition.

The following straightforward properties of parallel composition are important:

Lemma 26. *Given two compatible drags* $D = \langle V, R, L, X, S\rangle$ *and* $D' = \langle V', R', L', X', S'\rangle$,

(1) $D \oplus D'$ *preserves indegrees of* D *and* D' *at all vertices;*
(2) *the injections from* V *and* V' *to* $V \cup V'$ *are monomorphisms from* D *and* D' *to* $D \oplus D'$.

Proof. Preservation of indegrees at all vertices follows from the definitions of compatibility and sum. Being identities, these injections are injective morphisms that protect equimorphic subdrags and indegrees, hence are monomorphisms by Lemma 20. ☐

Parallel composition allows to define the intersection of two compatible drags:

Definition 27 (Intersection)**.** *The* intersection *of two compatible drags* D *and* D' *with vertices* V *and* V' *respectively is the subdrag of* $D \oplus D'$*, denoted* $D \cap D'$*, generated by* $V \cap V'$.

Once more, we remark that indegrees are preserved at all vertices of the intersection.

6.2. **Wiring.** The purpose of wiring a drag D is to add new edges to a drag by *connecting* sprouts to roots. Informally, a set of wires will be a set of pairs made out of a sprout and a root, written as $s \rightsquigarrow r$. Wiring D will be the action of redirecting all edges $u \longrightarrow s$ in D, including the roots of s, so that they become edges $u \longrightarrow r$ or roots of r in a new drag D'. Wiring may use a succession of wires like $s \rightsquigarrow t$ and $t \rightsquigarrow r$ that generate chains of wirings.

Definition 28 (Wire, origin, target)**.** *Given a drag* $D = \langle V, R, L, X, S\rangle$*, a* wire *is a pair* $s \rightsquigarrow r$ *of vertices of* D*, whose* origin s *is a sprout and* target r *a vertex different from* s.

Definition 29 (Wiring chain)**.** *Given a set of wires* W *of a drag* $D = \langle V, R, L, X, S\rangle$*, we define* $s >_W r$ *for* $s \in S$ *and* $r \in R$*, if* $s \rightsquigarrow r \in W$ *or if there is a vertex* t *such that* $s \rightsquigarrow t \in W$ *and* $t >_W r$.

In a wiring chain $s_0 \rightsquigarrow s_1 \rightsquigarrow \cdots \rightsquigarrow s_n \rightsquigarrow r$, all but possibly the final element r are sprouts, and the final element r is a root—and possibly also a sprout.

Definition 30 (Well-behaved set of wires)**.** *A finite set W of wires of D is* well-behaved *if:*

1. *functionality:* $\forall s \rightsquigarrow r, s \rightsquigarrow r' \in W : r = r'$*;*
2. *injectivity:* $\forall r \in R : \Sigma_{s >_W r}\, pred(s) \leq R(r)$*;*
3. *well-foundedness: W does not induce a cycle among the sprouts of D, that is, the restriction of $>_W$ to $S \times S$ is acyclic.*

The domain $\mathcal{D}om(\xi)$ of W is the set of sprouts that are origins of a wire.

Condition (1) implies that W is a partial function from S to V. We will therefore be able to consider the *restriction* of that function to a subset of its domain. If $V' \subseteq V$, we will say that W *restricts* to V' if $\forall s \rightsquigarrow r \in W : r \in V'$ if $s \in V'$.

Condition (2) means that vertex r is a root with a multiplicity large enough so that rewiring edges from $s_1, \ldots s_n$ to r does not require more roots of r than are available, which is yet another manifestation of multi-injectivity. In contrast with [DJ19], sprouts at the origin of a wire disappear with their roots, which are therefore lost if there were any.

Condition (3) allows us to compare sprouts. Well-foundedness of this order aims at defining wiring by induction.

It also implies the following:

Proposition 31. *The relation $\geq_W$ is a partial order for any well-behaved set of wires W.*

An empty set of wires is trivially well-behaved. A wire $s \rightsquigarrow t$, considered as a singleton set, is well-behaved iff it satisfies injectivity, that is, if the indegree of s is no larger than the number of roots of r.

We now define the drag obtained by adding wires to an existing drag, starting with the case of a single wire $s \rightsquigarrow r$. The idea is that sprout s is removed, all edges ending up in s (but not its roots) are moved to r using its roots in the same number:

Definition 32 (Elementary wiring)**.** *Given a drag $D = \langle V, R, L, X, S\rangle$ and a wire $s \rightsquigarrow r$, we define the drag $D_{s \rightsquigarrow r}$, after* wiring*, as the drag $\langle V', R', L', X', S'\rangle$ such that:*

1. $V' = V \setminus s$*;*
2. $R' = R \setminus (R(s) \cup R(r)) \cup r^{R(r) - pred(s)}$*;*
3. $L' = L \restriction V'$*, the restriction of labels L to vertices in V';*
4. $X' = X \setminus \{v \longrightarrow s : vXs\} \cup \{v \longrightarrow r : vXs\}$*;*
5. $S' = S \setminus s$*.*

In Definition 28, the condition that the origin of a wire is distinct from its target ensures that the sprout origin of the wire has disappeared from the resulting drag. The calculation of the new multiset of roots in item (2) expresses the fact that each edge redirected from the origin to the target of the wire consumes a root of the target, while the roots of the origin are lost.

In the particular case where $s : x$ and $t : y$ are both rootless isolated sprouts, wiring allows one to rename the sprout labeled x by the sprout labeled y.

Lemma 33. *Elementary wiring preserves indegree at all remaining vertices.*

Proof. Using the notations of Definition 32, the property is trivial for all vertices but r, and true for r since the removed roots of r are replaced by an equal number of predecessors of s. □

To define wiring for an arbitrary set of well-behaved wires, we write a non-empty well-behaved set of wires W as the union $s \rightsquigarrow r \cup W'$, where sprout s is maximal in $>_W$. Well-foundedness of $>_W$ allows us to wire $s \rightsquigarrow r$ first, and then recur on W', which can be easily shown well-behaved:

Lemma 34. *Let $W = s \rightsquigarrow r \cup W'$ be a well-behaved set of wires of D such that s is maximal in $>_W$. Then, W' is a well-behaved set of wires of $D' = D_{s \rightsquigarrow r}$.*

Proof. By maximality assumption, s does not occur in W' which is therefore a set of wires of D'. Functionality and membership follow straightforwardly from well-behavedness of W, as well as well-foundedness, since it restricts to subsets. Injectivity holds at all vertices since $\Sigma_{t >_W r} pred(t) = \Sigma_{t >_{W'} s} pred(t) + pred(s)$. □

It follows that all subsets of a well-behaved set of wires are well-behaved.

We can now define recursively wiring with an arbitrary well-behaved set W of wires. Not only do we define the new drag D_W, but also trace the vertices of the original drag along the recursive computation, with or without their root multiplicity.

Definition 35 (Wiring, resolution, natural injection)**.** *Given a well-behaved set of wires W of a drag D, we define the drag D_W, after wiring, by induction on the size of W, where for a non-empty set of wires W, we write $s \rightsquigarrow r \cup W'$ for $W = \{s \rightsquigarrow r\} \cup W'$,*

$$D_\varnothing = D$$
$$D_{s \rightsquigarrow r \cup W'} = (D_{s \rightsquigarrow r})_{W'}.$$

The resolution *$W(t)$ of sprout t of D in the domain of W is the (unique) root r in D that is the minimal one such that $t \geq_W r$, together with its root multiplicity in D_W* after *wiring. (There is a unique minimum on account of functionality of well-behaved sets of wires.) Also, the* natural injection *of D into D_W is the map $W_0(t)$ which returns the same vertex as $W(t)$ without its root multiplicity.*

Since W' is a well-behaved set of wires for the drag D_W by Lemma 34, the recursive call $(D_{s \rightsquigarrow r})_{W'}$ makes sense. The functions, resolution and natural injection, could also be defined by induction in the same way that the post-wiring drag was. Note that in case t is an internal vertex, the vertex itself is not changed by wiring, but its multiplicity might be.

Note also that a cycle is generated in a wired drag in case a sprout s is accessible in the original drag from its resolution, implying that a loop (a cycle of length 1) can only be generated on an internal vertex.

Example 36. *Figure 7 displays two examples of wiring, illustrating the recursive calculation of the result. We start with a drag union of two drags, and a well-behaved set W of two wires written underneath. The number of times a vertex is a root is indicated next to the root arrow's origin; the number 1 is often omitted. Labels can serve here as vertex names since no two vertices have the same label. The middle drag is the result of the first step of the calculation, with the remaining set of one wire written again underneath.*

For the first calculation, both edges that ended up in sprout x, which has disappeared, are now redirected to sprout y, which has a single root left, since two roots have been utilized by redirecting the edges $f \longrightarrow x$ and $h \longrightarrow x$ and one for the root of x. The rightmost drag is the final result, there is no set of wires left. Note

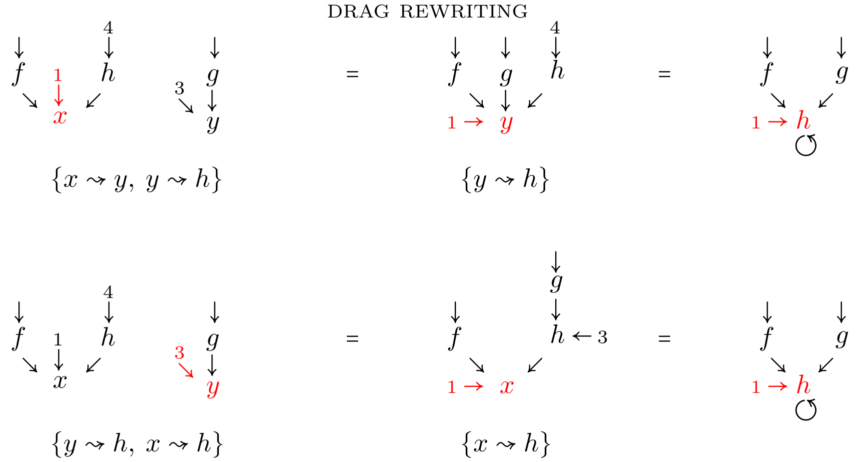


FIGURE 7. Formation of cycles via composition, two versions.

that starting with the wire $y \rightsquigarrow h$ in which y is not maximal would not make sense, since $x \rightsquigarrow y$ would not be a wire anymore after the first step of the calculation. The tracing of $W(x)$ is indicated in red, x moving to y and then to h.

The second calculation is similar. We note that it yields the same result. This is no coincidence, as we shall see. □

For the recursive calculation of Definition 35 to make sense, we need to show that the resulting drag does not depend upon the choice of a maximal wire $s \rightsquigarrow r$:

Lemma 37. *Given a drag $D = \langle V, R, L, X, S\rangle$ and a well-behaved set of wires W, D_W and $W(v)$, for each vertex $v \in V$, are well-defined.*

Proof. By induction on the size of W. We give the proof for the wiring definition.

If W is empty, the result is straightforward. If it contains a single maximal wire, the induction hypothesis applies. Otherwise, let $W = s \rightsquigarrow r \cup s' \rightsquigarrow r' \cup W'$ where $s \rightsquigarrow r$ and $s' \rightsquigarrow r'$ are distinct wires, maximal for $>_W$. Let $D' = (D_{s\rightsquigarrow r})_{s'\rightsquigarrow r'}$ and $D'' = (D_{s'\rightsquigarrow r'})_{s\rightsquigarrow r}$. Note that W' is well-behaved for both D' and D'', even if $r = r'$. By functionality, $s \neq s'$, and by maximality, $s' \neq r$ and $s \neq r'$; hence, redirecting the edges ending up in s to r and those ending up in s' to r' can be done in any order, showing that $D' = D''$.

Using Definition 35 twice for each calculation, $D_W = D'_{W'}$ if $s \rightsquigarrow r$ is chosen first, and $D_W = D''_{W'}$ if $s' \rightsquigarrow r'$ is chosen first. Since $D' = D''$, we conclude by the induction hypothesis that D_W is well-defined. □

Wiring has three important properties illustrated at Figure 7:

Lemma 38. *Let D be a drag and W a well-behaved set of wires of D. Then,*

1. *D_W and D have the same internal vertices and total number of edges—not counting roots as edges;*
2. *all vertices u of D_W have the same indegree in D and D_W;*
3. *whenever W' is a well-behaved set of wires such that $\forall t \in \mathcal{V}(D) : W(t) = W'(t)$, then $D_W = D_{W'}$.*

The fact that wiring preserves indegrees, just like monomorphisms must, will turn out to be a key observation (Lemma 88 below).

Proof. The third property follows from the other two, which hold because internal vertices, total number of edges, and indegree of vertices are all preserved by an elementary wiring step—thanks to Lemma 33 for indegree. □

6.3. **Coherence.** We haven't yet required that different wires sharing the same label satisfy an assumption implying that the corresponding sprouts are related. More precisely, given a drag D and a set of wires W, we want two sprouts $s : x$ and $t : x$ of D, with the same label x, to become equivalent after wiring. Ideally, we would like to check the property without computing D_W, that is, read it on D and W. In [DJ19], we required that $r = r'$ for any two wires $s : x \rightsquigarrow r$ and $s' : x \rightsquigarrow r'$ belonging to W, a model called *forced sharing*, and suggested that a more general equivalence should be drag isomorphism.

It turns out, however, that drag isomorphism is too weak. Take for example the drag $D = f(x, x, y, z, a, b)$ and the set of wires $W = \{x_1 \rightsquigarrow y, x_2 \rightsquigarrow z, y \rightsquigarrow a, z \rightsquigarrow b\}$. The sprouts x_1 and x_2 are replaced by equivalent drags, but won't remain equivalent in D_W since x_1 will be eventually replaced by a and x_2 by b. We could then expect that equimorphism, which is weaker than equality but stronger than isomorphism, could do.

And indeed, equimorphism in D of the subdrags generated by r and r' is one possible answer, although not the most general one. There is however a difficulty: Equimorphism is not preserved along the wiring process, although it is finally restored. For an example, consider the drag D made of three copies of $f(x)$, numbered 1,2,3. Let $W = x_3 \rightsquigarrow f_1 \cup W'$, with $W' = \{x_1 \rightsquigarrow f_2, x_2 \rightsquigarrow f_3\}$, be a set of wires that satisfies this equivalence condition. Wiring the sole wire $x_3 \rightsquigarrow f_1$ yields the drag D' made of three subdrags, the first two are still the same as before while the last has become $f_3(f_1(x_1))$. Wires W' no longer satisfy the property, since x_1 maps to f_2, which generates the subdrag $f_2(x_2)$, while x_2 maps to f_3, which generates now the subdrag $f_3(f_2(x_2))$, two non-isomorphic subdrags. On the other hand, applying all wires at once yields the drag which has three vertices f_1, f_2, f_3 and three edges $f_1 \longrightarrow f_2, f_2 \longrightarrow f_3, f_3 \longrightarrow f_1$. Obviously, the subdrags generated by f_1, f_2, f_3 are still equimorphic. This is the reason why we did not include coherence into the definition of a well-behaved set of wires: recurring on W would have become impossible. Wiring, hopefully, makes sense even when W is not coherent, hence our choice. The question however remains, whether we need a property weaker that equimorphism. To illustrate the need, let us consider the drag $D = f(x, x, y, a)$ and the set of wires $W = \{x_1 \rightsquigarrow y, x_2 \rightsquigarrow a, y \rightsquigarrow a\}$. Obviously, y and a do not generate equimorphic drags in D, but they do in D_W. This tells us that the condition should be checked on the result of wiring W in D. We now give the formal definition of a coherent set of wires:

Definition 39 (Coherence)**.** *Let D be a drag and W a well-behaved set of wires of D. We say that W is* coherent *(*strongly coherent*, respectively) if it satisfies the two following properties:*

1. *Fullness: All sprouts with the same label are the origin of a wire as soon as one is.*
2. *Equimorphism: For any two wires $s : x \rightsquigarrow r$ and $s' : x \rightsquigarrow r'$ of W, $W_0(r)$ and $W_0(r')$ generate equimorphic subdrags of D_W (of D, respectively).*

Note that forced sharing is a simple, particular case of strong coherence.

Like for the sum operation, wiring relates to the existence of certain morphisms between the wired drag and the original drag that will play a key rôle when it comes to rewriting drags.

Lemma 40. *Let D be a drag having no isolated sprout and W a well-behaved, coherent set of wires of D. Then the* natural injection *from D to D_W defines a monomorphism.*

Proof. The natural injection ι from D to D_W is a map ι that is the identity on the internal vertices of D, hence preserves their labels, is injective and indegree preserving by Lemma 38 (2), and maps every sprout to its resolution;

Since every vertex of D_W is a vertex of D, the set of entering edges is empty. Edges of D of the form $u \longrightarrow^i s_i$, where s_i is a sprout such that $v' = o(s_i)$, create the edge $o(u) \longrightarrow^i v'$ of D_W. This edge takes the place of a root at v in D if the inverse image of ι contains an internal vertex or is a sprout $s = o(s)$ shared between D and D_W, a root that has therefore disappeared in D_W, hence ensuring root preservation. The case where v' is a sprout whose inverse image is a set of sprouts all different from s is simply impossible here since a resolution vertex must be a vertex of D. We are left with showing that ι preserves equimorphic subdrags of D_W, which follows from coherence of W. □

Coherence can actually be checked without requiring the full computation of D_W: any property implying coherence and preserved by wiring would do. This is of course the case of forced sharing, but we can do better.

Lemma 41. *Given a drag D, a well-behaved set of wires W of D is coherent iff $W = W' \cup W''$ for some coherent W' and strongly coherent W''.*

In words, coherence is achieved in the drag resulting from the whole computation as soon as, for every variable x, the various wires $s : x \rightsquigarrow r$ generate equimorphic subdrags in the drag computed so far. Note that only nonlinear variables of D need be checked for strong coherence.

Proof. The only-if direction follows directly from the definition of coherence by taking $W'' = \varnothing$. The converse is by induction on the size of W'', the base case being obtained for $W'' = \varnothing$, which yields coherence of $W = W'$ by assumption.

If W'' is non-empty, let x be a variable maximal in $>_{W''}$ and $W'' = W^x \cup W'''$, where $W^x = \{s_i : x \rightsquigarrow r_i\}$ is the set of wires in W'' whose origins are labeled x. The origins of wires in W''' are therefore labeled by variables other than x. We will first show that (i) $W' \cup W^x$ is coherent, then that (ii) W''' is strongly coherent with respect to $D_{W' \cup W^x}$, before concluding that W is coherent via the induction hypothesis.
(i) Since x is maximal, the subdrags generated by the r_i's are identical in $D_{W'}$ and $(D_{W'})_{W^x}$, establishing property (i).
(ii) Since replacing sprouts labeled x in equimorphic subdrags of $D_{W'}$ by equimorphic subdrags of $D_{W'}$ yields equimorphic subdrags of $(D_{W'})_{W^x}$, property (ii) follows. □

6.4. **Product.** While wiring operates on a single drag, product operates on a pair of drags via a connecting device we call a *switchboard*:

Definition 42 (Switchboard)**.** *Given two disjoint drags D, D', a* switchboard ξ *for D, D' is a pair of partial functions $\langle \xi_D : \mathcal{S}(D) \to \mathcal{R}(D'), \xi_{D'} : \mathcal{S}(D') \to \mathcal{R}(D) \rangle$,* *called* switchboard components*, such that $\xi_D \cup \xi_{D'}$ is a coherent well-behaved set of wires for $D \oplus D'$. We also say that $\langle D', \xi \rangle$ is an* extension *of D and D' its* context extension*. Switchboard ξ is* one-way *if either one of ξ_D and $\xi_{D'}$ has an empty domain.*

Drags D and D' being disjoint, $\xi_D \cup \xi_{D'}$ is well defined. Therefore, ξ can be identified with $\xi_D \cup \xi_{D'}$. Note further that ξ_D and $\xi_{D'}$ need not be true injective functions as in [DJ19], where roots were lists with repetitions: Injectivity has been adapted to multisets of roots in our definition of a well-behaved set of wires.

Composition can now be defined as a wiring operation on $D \oplus D'$:

Definition 43 (Composition)**.** *Let $D = \langle V, R, L, X, S \rangle$ and $D' = \langle V', R', L', X', S' \rangle$ be disjoint drags, and ξ a switchboard for D, D'. Their* cyclic composition *or* product *is the drag $D \otimes_\xi D' = (D \oplus D')_\xi$.*

Example 44. *In Example 36, the first drag is the union of two drags that share no vertices, and the set of wires is just their switchboard. The result is therefore the composition of these two drags with respect to that switchboard.* □

Lemma 38(1) applies to composition: The total number of edges of $D \otimes_\xi D'$ is the sum of the number of edges of D and D', a property already noted in [DJ19]. The indegree is preserved at all vertices of $D \otimes_\xi D'$ since indegrees are preserved by the sum of disjoint drags and by wiring.

When restricted to one-way switchboards, cyclic composition is dubbed "sequential" in [Gad07]. Many other names coexist that target other classes of graphs and particular cases of composition.

A direct definition of a switchboard and of composition of two disjoint drags connected by a switchboard was given in [DJ19]. Our definition here assumes (a) that roots are multisets, instead of the lists there, (b) that resolution vertices have enough roots for transferring the edges of its corresponding origins, instead of edges and roots there, and (c) that coherence is ensured via equimorphism instead of sharing. These two models therefore have different behaviors.

Apart from these differences, both definitions are similar: our goal now is to give a definition of composition via wiring in the style of the direct definition used in [DJ19]. In the context of composition of two drags D, D' with respect to switchboard ξ, we define:

Definition 45 (Resolution)**.** *Let $D = \langle V, R, L, X, S \rangle$ and $D' = \langle V', R', L', X', S' \rangle$ be disjoint drags, and ξ a switchboard for D, D'. The* resolution *of a sprout $s \in S \cup S'$ is the vertex $\xi^!(s) = \xi_0(s)$, viewing the switchboard ξ as the set of wires W in Definition 35.*

The direct definition of composition has now become a simple property of the wiring definition:

Lemma 46 (Composition)**.** *Let $D = \langle V, R, L, X, S \rangle$ and $D' = \langle V', R', L', X', S' \rangle$ be compatible drags, and ξ be a switchboard for D, D'. Then $D \otimes_\xi D' = \langle V'', R'', L'', X'', S'' \rangle$, where*

(1) $V'' = (V \cup V') \smallsetminus \mathcal{D}om(\xi)$;
(2) $S'' = (S \cup S') \smallsetminus \mathcal{D}om(\xi)$;
(3) $R''(v) = \begin{cases} R(v) - \Sigma_{\xi^!(w)=v \wedge w \neq v} pred(w, D) & \text{if } v \in R \smallsetminus \mathcal{D}om(\xi) \\ R'(v) - \Sigma_{\xi^!(w)=v \wedge w \neq v} pred(w, D') & \text{if } v \in R' \smallsetminus \mathcal{D}om(\xi); \end{cases}$

(4) $L''(v) = \begin{cases} L(v) & \textit{if } v \in V \cap V'' \\ L'(v) & \textit{if } v \in V' \cap V''; \end{cases}$

(5) $X''(v) = \begin{cases} \xi^!(X(v)) & \textit{if } v \in V \setminus S \\ \xi^!(X'(v)) & \textit{if } v \in V' \setminus S'. \end{cases}$

The calculation of the multiset of roots of the composition is based on the preservation of indegrees by wiring, making it very simple: Edges accumulate along the computation until the very end, at which point they must be compensated for by the roots of the resolution.

Proof. By Definition 43, $D \otimes_\xi D' = (D \oplus D')_\xi$. We therefore show that $(D \oplus D')_\xi$ is the drag obtained from $D \oplus D'$ by (i) removing from the set of vertices all sprouts in $\mathcal{D}om(\xi)$, (ii) redirecting all edges ending up in a removed sprout to its associated resolution, and (iii) keeping the indegree unchanged for all vertices in $(V \cup V') \setminus \mathcal{D}om(\xi)$, which determines their number of roots as stated. The proof is by induction on the size of ξ considered as a set of wires. There are two cases two consider:

– In the empty case, $D \otimes_\varnothing D' = D \oplus D'$. Then, properties (i), (ii), and (iii) hold trivially.

– Otherwise, by Lemma 37, we can choose any wire $s \rightsquigarrow r$ such that $\xi = W \cup s \rightsquigarrow r$, where s is maximal in ξ; hence $D \otimes_\xi D' = ((D \oplus D')_{s \rightsquigarrow r})_W$. We then conclude by the induction hypothesis, since W has one wire fewer than does ξ, that $D \otimes_\xi D'$ is obtained from $(D \oplus D')_{s \rightsquigarrow r}$ as claimed. Since s is maximal, it follows from the definitions of resolutions $\xi_0(x)$ and $\xi^!(x)$ that $(D \oplus D')_\xi$ is obtained from $D \oplus D'$ as claimed. □

Not all vertices of the composition $D \otimes_\xi D'$ may be reached via ξ from the sprouts of one of them. We denote by $\xi(D)$ the subdrag of D' generated by the vertices in $\xi_D(\mathcal{D}om(\xi_D))$, which contains all vertices of D' which are reached from sprouts of D.

There are important particular cases of switchboards that impose additional, hence stronger, coherence conditions:

- *Equality switchboard*: $\forall s \neq t \in S \cup S'$ such that $L(s) = L(t)$, we have $\xi(s) = \xi(t)$.
- *Inequality switchboard*: $\forall s \neq t \in S \cup S'$ such that $L(s) = L(t)$, we have $(D \oplus D')|_{\xi(s)}$ and $(D \oplus D')|_{\xi(t)}$ have no vertex in common.

Composition is said to *force sharing* if ξ is a switchboard with equality, and to *force cloning* if ξ is a switchboard with inequality. Term rewriting is based on cloning switchboards while dag rewriting is based on equality switchboards. Our notion of composition can potentially do both, and can also achieve partial sharing by having $(D \oplus D')|_{\xi(s)}$ and $(D \oplus D')|_{\xi(t)}$ sharing subdrags, in particular sprouts, when possible. These question will be studied in more detail in Section 11.

Notations: Given two drags D and D' and a switchboard $\xi = \{s_i \rightsquigarrow r_i\}_i$ for (D, D'), we will sometimes need to restrict the switchboard ξ to some subsets of vertices V of $\mathcal{V}(D)$ and V' of $\mathcal{V}(D')$. We will therefore denote by $\xi_{V \to V'}$ the restriction of switchboard ξ to the set of wires $\{s_i \rightsquigarrow r_i : s_i \in V, r_i \in V'\}_i$. We will use $\xi_{V,V'}$ for the switchboard $\xi_{V \to V'} \cup \xi_{V' \to V}$, $\xi_{V \to}$ for the switchboard $\xi_{V \to \mathcal{V}(D')}$, and $\xi_{\to V'}$ for the switchboard $\xi_{\mathcal{V}(D) \to V'}$.

7. Decomposition of Drags

We now investigate to what extent a drag can be expressed in terms of simpler drags by means of sum and product so as to obtain a *drag expression*:

Definition 47 (Drag expression)**.** *By a* drag expression*, we mean any expression built from a given set of* drag components $\{D_i\}_i$ *such that* D_i *and* D_j *share no vertex nor variable if* $i \neq j$*, by means of sum and product of drags. Drags occurring in a drag expression are its* drag components*. A drag* D *is a* trivial *drag expression whose drag* D *is its only drag component.*

Note that product alone would suffice to build drag expressions, writing a sum $D \oplus D'$ as the product $D \otimes_{\varnothing} D'$.

An initial, straightforward answer is that we can decompose a drag according to some subset of its internal vertices by using the corresponding subdrag and associated context:

Lemma 48 (Reconstruction)**.** *Given a drag* D *and a subset* W *of its vertices, then* $D = D|_W \otimes_\xi D\upharpoonright_W$ *for some switchboard* ξ*.*

Proof. Using the notations of Definition 8, it suffices to define ξ, which maps every new sprout s_v of the restriction (of the context, respectively) to the vertex v of the context (of the restriction, respectively). The equality claim then follows easily using preservation of indegrees by wiring. □

We will indeed use the restriction of D to some carefully-chosen single internal vertex, give a specific definition for that case, and treat some special cases separately.

First, we need to define what are "atomic" drags, the kind we like to have in a full decomposition, and the non-atomic ones that we want to eliminate:

Definition 49 (Connected drag)**.** (1) *A* connected *drag is any drag* $\langle V, R, L, X, S\rangle$ *whose set of vertices is generated by successor and equal labeling of sprouts, that is, any subset* W *of* V *closed under these two operations must be* V *itself:*

$$\forall W \subseteq V(\forall u \forall v \in V(uXv : u \in W \text{ iff } v \in W) \text{ and } \forall s : x \forall t : x \in V(s \in W \text{ iff } t \in W)) \Rightarrow W = V$$

(2) *A* flat *drag is a connected drag with no non-trivial path between internal vertices.*

(3) *An* atomic vertex *is a drag with only a single internal vertex and any number of roots and different sprouts as successors, all with different labels.*

(4) *A nonempty set of pairwise distinct sprouts is* atomic *if they all share the same variable, each with any number of roots.*

(5) *An* atomic drag *is an atomic vertex or an atomic set of sprouts.*

For example, the drag with two internal vertices sharing one sprout is flat, while the drag made of a single loop on an internal vertex is not. Note also that the drag made of two drags $f(s : x)$ and $g(t : x)$ is connected by our definition, as if s and t where the same shared sprout.

A major property of non-flat connected drags is that they all have non-necessarily distinct internal vertices u, v such that u is a predecessor of v.

Atomic drags are of course flat, but there are also flat non-atomic drags. On the other hand, non-connected drags can always be written as a sum of connected drags. We therefore consider the decomposition of non-flat connected drags first, before addressing the case of flat non-atomic connected drags.

In what follows, we give a sequence of four transformation rules in the form of as many lemmas. These transformations take as input a drag expression E containing a drag component D that is not yet atomic, but instead: (i) comprises pairwise distinct connected components; or (ii) is a non-flat connected component with an edge $u \longrightarrow v$ between two distinct internal vertices; or (iii) is a non-flat connected component with a loop $u \longrightarrow u$ on some internal vertex u; or else (v) is a flat connected component with sprouts $s_i : x$ that are either shared or sharing the variable label x, or both. Clearly, any non-atomic drag belongs to one of these four categories. Therefore, applying these transformations repeatedly to a not-yet atomic drag component D of E will eventually transform E into a drag expression E' all of whose components are atomic drags. This is so because the drag expressions E' obtained from E are simpler than E, in some well-founded order, hence implying that any sequence of transformation is finite.

Before specifying the transformation rules, we define the order used to compare drag expressions:

Definition 50. *To a given drag D, we associate the triple $\langle \#I, In, M \rangle$, where*

1. *$\#I$ is the number of its internal vertices plus the edges between them;*
2. *In is the multiset of its sprouts' indegrees;*
3. *M is the multiset counting, for each variable $x \in \mathcal{V}ar(D)$, the number of its sprouts that are labeled x.*

Denoting by $>_{\mathbb{N}}$ the usual order on natural numbers, triples—hence drags—are compared in the well-founded order $\gg = (>_{\mathbb{N}}, >_{\mathbb{N}}^{mul}, >_{\mathbb{N}}^{mul})^{lex}$, where ^{lex} and ^{mul} denote lexicographic and multiset extensions of an order, respectively. Drag expressions, interpreted as the multiset of interpretations of their drag components, are compared in the well-founded order $\gg^{mul}$.

Lemma 51. *Let D be a drag made of pairwise distinct connected drags $D_1, \dots, D_n$, with $n > 1$. Then, $D = D_1 \oplus \cdots \oplus D_n$ is a drag expression such that $\forall i : D \gg D_i$.*

Proof. The only difficulty is the ordering statement. If there are two or more D_i's with internal vertices, the result is clear. If all D_i's are made of sprouts, their first component is 0, as for D, but the second has fewer 0's than in D, hence decreases strictly. If, say, D_1 has at least one internal vertex but all other D_i's do not, then D has strictly more internal vertices than the D_i's, while D_1 has the same number of them as does D. But its multiset of sprout indegrees must have decreased strictly. □

Lemma 52 (Decomposition)**.** *Given a non-flat connected drag $D = \langle V, R, L, X, S \rangle$, let v be an internal vertex of D of indegree p, labeled f of arity n, having at least one predecessor $u \neq v$, and whose successors in D are the vertices $v_1, \dots, v_n$. Then—denoting v by f:*

$$D = D_f \otimes_\xi D' \text{ with } \xi = \{s \rightsquigarrow v, s_1 \rightsquigarrow w_1, \dots, s_n \rightsquigarrow w_n\}$$

is a drag expression such that $D \gg D_f$ and $D \gg D'$, where:

1. *$D_f = f^{[p]}(s_1 : x_1, \dots, s_n : x_n)$;*
2. *$D' = \langle V', R', L', X', S' \rangle$;*
3. *$V' = (V \setminus v) \cup s$, where s is a fresh sprout;*

(4) $\forall u \in V \setminus (v \cup \{v_i\}) : R'(u) = R(u)$; $\forall u \in \{v_i\}_i : R'(u) = R(u) + 1$; $R'(s) = 0$;
(5) $\forall u \in V \setminus v : L'(u) = L(u)$; $L'(s) = x$, *where* x *is a fresh variable;*
(6) $\forall u, w \in V \setminus s : uX'w$ *iff* uXw; $\forall u \in V \setminus s : uX's$ *iff* uXv;
(7) $S' = S \cup s$;
(8) $w_i = s$ *if* $v_i = v$, *and otherwise it is* v_i.

Proof. Note that D is a non-flat connected drag, since $u \longrightarrow v$. The switchboard is designed so that it is trivially coherent and well-behaved, and D is reconstructed from two drag components sharing no vertex or variable. We do not and need not assert that D' itself is a connected drag; it may not be if the subdrag generated by some v_i has no shared vertex with its associated context drag.

The equality claim follows from preservation of indegrees by composition.

Drag D' is smaller than D since an internal vertex has been removed, and the edges between the remaining internal vertices are those of D.

For the last claim, notice that the drag $f^{[p]}(s_1 : x_1, \ldots, s_n : x_n)$ has a single internal vertex v and no edge $v \longrightarrow v$. □

Example 53. *Consider a drag* D*, reduced to two internal vertices labeled* g *and* a*, of arities 1 and 0, respectively, plus a single edge* $g \longrightarrow a$*. We get* $D = a^{[1]} \otimes_{s \mapsto a} g(s^{[0]})$. □

Lemma 54 (Loop decomposition)**.** *Let* D *be a drag with an internal vertex* v *of indegree* p*, labeled* f *of arity* n*, whose successors in* D *are the vertices* $v_1, \ldots, v_n$*, with* $v_i = v$ *for some* i*. Then,*

$$D = D' \otimes_\xi t^{[1]} : y \text{ with } \xi = \{s \rightsquigarrow t, t \rightsquigarrow v\}$$

is a drag expression such that $D \gg D'$ *and* $D \gg t^{[1]}$*, where:*

(1) $D' = \langle V', R', L', X', S' \rangle$;
(2) $V' = V \cup s$, *where* s *is a fresh sprout;*
(3) $\forall u \in V \setminus v : R'(u) = R(u)$; $R'(v) = R(v) + 1$; $R'(s) = 0$;
(4) $\forall u \in V : L'(u) = L(u)$; $L'(s) = x$, *where* x *is a fresh variable;*
(5) $\forall u, w \in V : uX'w$ *iff* uXw *except for the edge* $v \longrightarrow^i v$ *of* D *replaced by* $v \longrightarrow^i s$ *in* D';
(6) $S' = S \cup s$.

This lemma allows us to eliminate one loop at a time. When there are several loops on the internal vertex v, they have to be eliminated one by one. We could of course give a more general statement eliminating them all at once, at the price of a slightly more complicated statement and proof.

Proof. Similar to the proof of Lemma 52, except for the ordering statements.

Here D' has the same total number of internal vertices but strictly fewer edges between them since some edge $v \longrightarrow v$ of D has been replaced by an edge $v \longrightarrow s$ in D', which does not count because s is a sprout. As for the other drag, it has no internal vertices, while D has at least one. □

Example 55. *Consider a drag* D *reduced to a single internal vertex* v *labeled* f *of arity 1, and a single looping edge* $f \longrightarrow f$*. Then, we have* $D = f^{[1]}(s : x) \otimes_{s \mapsto t, t \mapsto f} t^{[1]}$. □

We next consider the case of non-atomic connected flat drags. They are made of one or more internal vertices connected via their successor sprouts, some of them being shared, or sharing the same variable label, or both. We will eliminate at once all connections related to a given variable label. If there are several variable labels involved in the connections, they will have to be eliminated one by one.

Lemma 56 (Unsharing decomposition)**.** *Let D be a connected flat drag whose nonempty set $\{s_i : x\}_{i=1}^n$ $(n \geq 1)$ of distinct sprouts sharing variable label x, having q_i predecessors and indegree $p_i \geq q_i$, respectively, contains at least two sprouts, or one shared sprout, or both. Then*

$$D = D' \otimes_\xi \left(s_1^{[p_1]} \cdots s_n^{[p_n]} \right)$$

is a drag expression such that $D \gg D'$ and $D \gg s_1^{[p_1]} \dots s_n^{[p_n]}$, where:

(1) *D' is obtained from D by replacing each vertex s_i by fresh rootless sprouts $t_{i,1} : y_{i,1}, \dots, t_{i,q_i} : y_{i,q_i}$ and every edge $v \longrightarrow s_i$, if any, by edges $v \longrightarrow t_{i,j}$, with $1 \leq j \leq q_i$;*
(2) *$\xi = \{t_{i,k} \rightsquigarrow s_i\}_i$.*

Note that the drag $s_1^{[p_1]} \cdots s_n^{[p_n]}$, all of whose sprouts share the same variable, is an atomic variable drag that cannot be decomposed any further without violating our notion of drag expression, since all these sprouts share label x.

Proof. In case $p_i = 0$, the switchboad ξ maps a rootless sprout t_{i_1} to a rootless sprout s_i. Since all sprouts labeled x, whether shared or nor, are renamed with the appropriate number of fresh sprouts, the resultant product is a drag expression. The equality statement is again a straightforward consequence of indegree preservation.

Finally, D' has the same number of internal vertices as D. If $n > 1$, the number of sprouts labeled x decreases strictly while the multiset of sprout indegrees does not increase. Or else $p_1 > 1$ and the multiset of sprouts indegrees decreases strictly. We are left with the case of a flat drag with no internal vertices but several sprouts labeled x. In that case, D and D' have the same first two components, but the third decreases strictly. The other ordering statement is straightforward. □

Here are two examples of flat non-atomic drags decomposed into atomic drags:

$$f(s^{[0]} : x, t^{[0]} : x) = f(s'^{[0]} : y_1, t'^{[0]} : y_2) \otimes_{s' \rightsquigarrow s, t' \rightsquigarrow t} (s^{[1]} : x \quad t^{[1]} : x)$$
$$f(s^{[0]} : x, s^{[0]} : x) = f(s' : y_1, t' : y_2) \otimes_{s' \rightsquigarrow s, t' \rightsquigarrow s} s^{[2]}$$

—assuming now that $s^{[0]}$ is shared.

Note the difficulty in representing the drag of the second example faithfully by means of the expression $f(s^{[0]} : x, s^{[0]} : x)$. We have to make explicit that $s^{[0]}$ is shared. On the other hand, our product-based representation is faithful; sharing is attained by calculating the drag expression. Note that this representation avoids the kind of unary binding notation usual in programming languages: The annotated product operation is a binder for its arguments.

The decomposition properties can be used to decompose a given drag D into atomic drags, so as to obtain a drag expression built from atomic drags by means of sum and product. Initially, we are given a drag D, considered as a (trivial) drag expression. We get the following:

Theorem 57. *For every drag D, there exists a drag expression made of atomic drags whose evaluation yields D.*

Proof. By its definition, the order on drag expressions is monotonic: replacing in E a drag component D by a drag expression D' all of whose components are strictly smaller than D yields a strictly smaller drag expression E'.

Since every non-atomic dag is either made of several distinct connected components, or is a non-flat connected component, or is a flat connected component that is not yet atomic, all cases have been taken care of. Therefore, by applying Lemmas 51, 52, 54, and 56 for as long as possible—termination being ensured by the well-founded order on drag expressions, we get a drag decomposition into atomic components. □

A drag can therefore be defined by induction from atomic pieces glued together by appropriate compositions, which gives rise to an (non-unique) algebraic notation for drags.

Example 58. *Let f be of arity 2 and h of arity 1. The connected drag D with vertices f_1, f_2 labeled f, vertices h_1, h_2, h_3 labeled h, and edges $f_1 \longrightarrow^1 f_2, f_1 \longrightarrow^2 h_2, f_2 \longrightarrow^1 h_1, f_2 \longrightarrow^2 h_2, h_1 \longrightarrow^1 h_1, h_2 \longrightarrow^1 h_3, h_3 \longrightarrow^1 f_2$ has as its drag expression*

$$f_2^{[2]}(x_1, x_2) \otimes_{x_1 \rightsquigarrow h_1, x_2 \rightsquigarrow h_2, x_3 \rightsquigarrow f_2} \Big(f_1(x_3, h_2^{[1]}(h_3(x_3))) \oplus h_1^{[1]}(\textit{SELF})\Big)$$

where x_3 denotes a shared sprout, obtained by applying successively Lemma 52 to the vertex f_2 and then Lemma 51 to the resultant drag. We continue now with the right-hand side argument of $\otimes$. Applying first Lemma 52 at vertex h_2, we get:

$$(h_2^{[2]}(x_4) \otimes_{x_4 \rightsquigarrow h_3, x_5 \rightsquigarrow h_2} (f_1(x_3, x_5) \quad h_3^{[1]}(x_3))) \oplus h_1^{[1]}(\textit{SELF})$$

Then applying Lemma 56 to the flat sub-expression sharing sprout s_3, which is the right-hand side argument of $\otimes$, yields the drag expression:

$$(h_2^{[2]}(x_4) \otimes_{x_4 \rightsquigarrow h_3 x_5 \rightsquigarrow h_2} ((f_1(x_6, x_5) \otimes_{x_6 \rightsquigarrow x_3, x_7 \rightsquigarrow x_3} h_3^{[1]}(x_7)))) \oplus h_1^{[1]}(\textit{SELF}))$$

And now applying Lemma 54 to the rightmost subdrag, being a loop, gives

$$(h_2^{[2]}(x_4) \otimes_{x_4 \rightsquigarrow h_3, x_5 \rightsquigarrow h_2} ((f_1(x_6, x_5) \otimes_{x_6 \rightsquigarrow x_3, x_7 \rightsquigarrow x_3} h_3^{[1]}(x_7)))) \oplus (h_1^{[2]}(x_6) \otimes_{x_6 \rightsquigarrow x_8 x_8 \rightsquigarrow h_1} x_8^{[1]}$$

a decomposition with no non-atomic components left. Putting the pieces together, we get the following drag decomposition of the starting drag:

$$\begin{array}{ll} f_2^{[2]}(x_1, x_2) & \otimes_{x_1 \rightsquigarrow h_1, x_2 \rightsquigarrow h_2, x_3 \rightsquigarrow f_2} \\ & \quad \big(h_2^{[2]}(x_4) \otimes_{x_4 \rightsquigarrow h_3, x_5 \rightsquigarrow h_2} \quad ((f_1(x_6, x_5) \otimes_{x_6 \rightsquigarrow x_3, x_7 \rightsquigarrow x_3} \quad h_3^{[1]}(x_7)))\big) \\ & \quad \oplus \\ & \quad (h_1^{[2]}(x_6) \otimes_{x_6 \rightsquigarrow x_8 x_8 \rightsquigarrow h_1} \quad x_8^{[1]})\big) \end{array}$$

The following decomposition of the same drag

$$\begin{array}{lll} f_1(x_1, x_2) & \otimes_{x_1 \rightsquigarrow f_2, x_2 \rightsquigarrow h_2} & \\ & \big(f_2^{[2]}(x_3, x_4) & \otimes_{x_3 \rightsquigarrow h_1, x_4 \rightsquigarrow h_2, x_6 \rightsquigarrow f_2} \\ & & \big(h_2^{[2]}(x_5) \otimes_{x_5 \rightsquigarrow h_3} h_3^{[1]}(x_6)\big) \quad \oplus \quad \big(h_1^{[1]}(x_7) \otimes_{x_7 \rightsquigarrow x_8, x_8 \rightsquigarrow h_1} (x_8)^{[1]}\big)\big) \end{array}$$

cannot be obtained by using our lemmas, since we are missing an analog of Lemma 52 applying to a vertex without ancestors of a non-flat drag. This additional decomposition lemma can be surmised by the reader; it would be needed in case we were interested in all possible decompositions of a given drag. Note that we end up here with a decomposition using again 8 fresh sprouts. This is no surprise: one sprout is needed for each incoming edge in the original drag, plus one for each loop, since *a loop decomposition requires two wires. The reader can verify that all switchboards used in these decompositions are well behaved and that the result is indeed the drag D in both cases.* □

8. Algebra of Drags

Once equipped with sum and product, drags enjoy a very rich algebraic structure which is known to be suitable for expressing distributed computations [MM90, MMS97].

Lemma 59. *Drag sum is associative, commutative, idempotent, and has the empty drag as an identity element.*

Noting that a drag is compatible with itself, all these properties are straightforward. We proceed with product:

Lemma 60. *Drag product is associative, commutative, and has an identity element, the empty drag. Other identities are isolated sprouts provided they belong to the domain of the switchboard, and do not belong to its image.*

Proof. Commutativity is straightforward here, because of the root structure as a multiset. The empty drag is again an identity for any drag D, since ξ must be empty, and therefore $D \otimes_{\varnothing} \varnothing = D \oplus \varnothing = D$. Let now $s^{[n]}$ be a drag reduced to a sprout which is not a vertex of D, and $s \rightsquigarrow v$ be a wire for (D, s). By assumption, s does not belong to the image of ξ_D. Then, a straightforward calculation shows that $D \otimes_{s \rightsquigarrow v} s^{[n]} = D$.

We are left with associativity. Consider the drag $(C \otimes_\xi D) \otimes_\zeta E$. We prove first that $\xi \cup \zeta$ is a well-behaved set of wires for $C \oplus D \oplus E$. By the definition of a switchboard, ξ and ζ are well-behaved sets of wires for $C \oplus D$ and $(C \otimes_\xi D) \oplus E$, respectively. Since ζ maps remaining sprouts of $C \oplus D$ after wiring with ξ to roots of E, and sprouts of E to remaining roots of $C \oplus D$ after wiring with ξ, $\xi \oplus \zeta$ is well-defined and satisfies functionality, coherence and injectivity. It is also well-founded, since chains of sprouts for $\xi \cup \zeta$ are either chains of sprouts for ξ or for ζ which both satisfy well-foundedness.

We construct now two new sets of wires, ζ' for $D \oplus E$, and ξ' for $C \oplus (D \otimes_{\zeta'} E)$ such that $\xi' \cup \zeta' = \xi \cup \zeta$, showing that $\xi' \cup \zeta'$ is a well-behaved set of wires for $C \oplus D \oplus E$, which implies that ξ' and ζ' are well-behaved sets of wires for $C \oplus (D \otimes_{\zeta'} E)$ and $D \oplus E$, respectively. We now classify each wire $s \rightsquigarrow r \in \xi \cup \zeta$ as a wire of ξ' or ζ':

(1) $s \in \mathcal{S}(C) : s \rightsquigarrow r \in \xi'$;
(2) $r \in \mathcal{R}(C) : s \rightsquigarrow r \in \xi'$;
(3) $s \in \mathcal{S}(D)$ and $r \in \mathcal{R}(E) : s \rightsquigarrow r \in \zeta'$;
(4) $s \in \mathcal{S}(E)$ and $r \in \mathcal{R}(D) : s \rightsquigarrow r \in \zeta'$.

The equality between both obtained drags now follows from routine calculations. □

We finally consider the distributivity law, important for distributed computations, that is (omitting switchboards), an equality of the form $C \otimes (D \oplus E) = (C \otimes D) \oplus (C \otimes E)$. There are two obstacles: The first is that the sum $(C \otimes D) \oplus (C \otimes E)$ must make sense, which requires that the compatibility of drags D and E, a reasonable assumption, is preserved by their product with C. Unfortunately, this requires the very strong assumption that there is no wire from C except those with heads at shared vertices of D and E. In case D and E are disjoint, no wire would be allowed from C, that is, $\xi_C = \varnothing$. The second obstacle is that wirings between D and E going through C in $C \otimes (D \oplus E)$ cannot be reproduced, in general, in $(C \otimes D) \oplus (C \otimes E)$, unless wiring again the result, which is not expected from a distributivity law whose rôle is to transform a product into a sum. Fortunately, the assumption that $\xi_C = \varnothing$ helps again. We therefore show the property below under this assumption which forbids the advent of new cycles by making a product, which is of course less restrictive than forbidding any cycle whatsoever.

Lemma 61. *Let D, E be compatible drags with shared subdrag G, E a drag disjoint from $D \oplus E$, and ξ a switchboard for $(D \oplus E, C)$, such that $\mathcal{I}ma(\xi_C) \subseteq \mathcal{V}(G)$. Then, drags $D \otimes_\xi C$ and $E \otimes_\xi C$ are compatible with shared subdrag $G \otimes_\xi C$, and*

$$(D \oplus E) \otimes_\xi C = (D \otimes_\xi C) \oplus (E \otimes_\xi C).$$

Proof. Our assumption that all wires whose origin is a sprout of C have their target in G and that all other wires have their origin in the context of G in $D \oplus E$, ensures that $G \otimes_\xi C$ is the shared subdrag of $D \otimes_\xi C$ and $E \otimes_\xi C$, implying compatible. Distributivity follows since there is no way to establish new edges between the contexts of G in D and E. □

Analyzing various counterexamples to distributivity in case the present assumptions are not met, we have observed that a more general compatibility property is needed instead of the present one, namely unifiability of D and E, their sum being their most general unifier. (On unification of drags, see [JO23].) This lead goes far beyond the objectives of the present framework, and is therefore left as a hint for motivated readers.

Drags enjoy a rich algebraic structure, which is known to be suitable for expressing distributed computations [MM90, MMS97].

9. Sharing Equivalence

Next, we define and study the equivalence on drags defined by sharing subdrags, which will play an important rôle for defining rewriting.

Definition 62 (Sharing)**.** *A drag is* maximally shared *if no two distinct subdrags are equimorphic.*

Note that a maximally shared drag must be linear.

First, we define the *maximally shared form* $D{\downarrow}$ of a drag D by iterating the following *sharing* transformation as long as necessary:

(1) Assume $E_1, \dots, E_n, F$ are all pairwise distinct maximally shared subdrags of D equimorphic to some subdrag F of D, called the *class* of F in D, and let C_F be the *context* of $\overline{F} = E_1 \oplus \cdots \oplus E_n \oplus F$. By Lemma 48, $D = C_F \otimes_\xi \overline{F}$ for some ξ. The class F is said to be *trivial* if it consists of the single drag F only ($n = 0$), and *nontrivial* otherwise ($n > 0$).

Claim 63. *Assume some drag in a class $\overline{E}$ is equimorphic to (possibly several) strict subdrags of a drag in a class $\overline{F}$. Then, all drags in the class $\overline{F}$ contain as a subdrag drags in the class $\overline{E}$. This shows that the well-founded subdrag order lifts to classes of drags. As a consequence, some classes are minimal in this order.*

(2) Assuming D is not maximally shared, there exists at least one minimal nontrivial class $\overline{F}$ in D. Let, therefore, $o_i : E_i \hookrightarrow F$ be the equimorphism

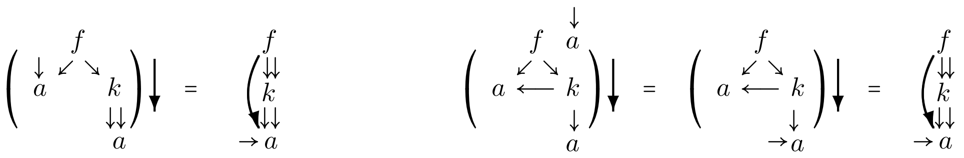


FIGURE 8. Maximally shared form of a drag.

from E_i to F and $o = \bigcup_i o_i$. We define $\xi' = o \circ \xi$ and $D' = C_F \otimes_{\xi'} F$. In words, D' is obtained from D by replacing any edge of D from an internal vertex u of C to a vertex v of some E_i by an edge from u to $o_i(v)$, and transfer any root of a vertex v of some E_j to $o_j(v)$, resulting in the drag D', which contains a unique element of the class of F, F itself. Note that this step does not create any new class of equimorphic drags. The choice of F in its class implies that the maximally shared form of D will be defined up to equimorphism.

Lemma 64. *The maximally shared form $D\!\downarrow$ of a drag D exists and is unique up to equimorphism.*

Proof. A sharing step strictly decreases the number of nontrivial classes of the drag D. It follows that it terminates, and therefore maximally shared forms exist. We prove uniqueness by showing that any two different sharing steps commute, and then conclude by the Diamond Lemma [Hin69].

Let $\overline{G}, \overline{H}$ be two different minimal classes of equimorphic drags of D that can be shared each in turn. By minimality of both classes, we can consider the context C of $\overline{G} \oplus \overline{H}$ such that $D = C \otimes_\xi (\overline{G} \oplus \overline{H}) = C \otimes_\xi (\overline{G} \otimes_\varnothing \overline{H})$. Using associativity and commutativity of product, we get $D = (C \otimes_{\xi_{C,\overline{H}}} \overline{H}) \otimes_{\xi_{C,\overline{G}}} \overline{G} = (C \otimes_{\xi_{C,\overline{G}}} \overline{G}) \otimes_{\xi_{C,\overline{H}}} \overline{H}$, showing that $(C \otimes_{\xi_{C,\overline{H}}} \overline{H})$ and $(C \otimes_{\xi_{C,\overline{G}}} \overline{G})$ are the contexts in D of $\overline{G}$ and $\overline{H}$, respectively. Let now $E = C \otimes_{\xi_{C,\overline{H}}} \overline{H}) \otimes_{\xi_{C,G}} G$ and $F = (C \otimes_{\xi_{C,\overline{G}}} \overline{G}) \otimes_{\xi_{C,H}} H$, obtained from D by sharing classes $\overline{G}$ and $\overline{H}$, respectively. Using associativity and commutativity again, we can now share the classes $\overline{G}$ and $\overline{H}$ in E and F, respectively. We get $E' = (C \otimes_{\xi_{E,H}} H) \otimes_{\xi_{C,G}} G$ and $F' = (C \otimes_{\xi_{C,G}} G) \otimes_{\xi_{C,H}} H$. Using associativity and commutativity again, we get $E' = (C \otimes_\xi (G \otimes_\varnothing H) = F'$. □

Example 65. *Figure 8 shows two examples of drags that have the same maximally shared form. For the left drag, the two subdrags reduced to a vertex labeled a are equimorphic. The maximally shared form is obtained in one step. For the right drag, two steps will be needed, as shown on the figure.* □

Lemma 66. *Given a drag D, there exists a morphism $o : D \to D\!\downarrow$, such that all vertices of D sent to the same vertex by o generate subdrags of D that are sharing-equivalent.*

Proof. Straightforward, noticing that each class of equimorphic drags in a drag has a representative (the one chosen by sharing) in normal form. □

Definition 67 (Sharing equivalence)**.** *Two drags are* sharing-equivalent *if they have equimorphic maximally shared forms.*

Simple consequences are that equimorphic drags are sharing-equivalent and that drags that are sharing-equivalent have the same sets of variables.

We now show that sharing-equivalence is closed by the operations on drags that we have defined. First, obviously,

Lemma 68. *Sharing-equivalence is closed under parallel composition.*

Definition 69 (Sharing equivalence of wires)**.** *Let D, D' be two drags that are sharing-equivalent. Two well-behaved set of wires $W = \{s_i \rightsquigarrow r_i\}_i$ of D and $W' = \{s'_j \rightsquigarrow r'_j\}_j$ of D' are* sharing-equivalent *if*

- *the sets of variables labeling the sprouts $\{s_i\}_i$ and $\{s'_j\}_j$ are identical;*
- *for all sprouts s_i, s'_j sharing the same label, $D\downharpoonright_{r_i}$ and $D'\downharpoonright_{r'_j}$ are sharing-equivalent.*

Two rewriting extensions $\langle E, \xi\rangle$ of D and $\langle E', \xi'\rangle$ of D' are sharing-equivalent *if so are E and E', and ξ and ξ'.*

The definition of sharing-equivalence for sets of wires makes sense since the same variables label the respective sprouts of sharing-equivalent drags. It makes sense for extensions by Lemma 68, since ξ and ξ' are well-behaved sets of wires of $D \oplus E$ and $D' \oplus E'$ by definition. Sharing-equivalence is thus an equivalence on drags, sets of wires, and extensions.

Lemma 70. *Given two sharing-equivalent well-behaved sets of wires ξ, ζ of a drag D, D_ξ and D_ζ are sharing-equivalent.*

Proof. By induction on the number of wires in ξ. If ξ is empty, then ζ must be empty too since they are sharing-equivalent, and the result holds in that case. Otherwise, neither ξ nor ζ can be empty. Let $s \rightsquigarrow r \in \xi$ be a maximal wire. By the definition of sharing-equivalence for sets of wires, there must exist a wire $s' \rightsquigarrow r' \in \zeta$ such that $D\downharpoonright_r$ and $D\downharpoonright_{r'}$ are sharing-equivalent, implying that neither are sprouts or both are in which case their label is the same. By coherence of a well-behaved set of wires, we can always choose $s = s'$. It follows that $s \rightsquigarrow r'$ must be maximal in ζ. The drags $D_{s\rightsquigarrow r}$ and $D_{s\rightsquigarrow r'}$ are clearly sharing-equivalent. Since $\xi \setminus s \rightsquigarrow r$ and $\zeta \setminus s \rightsquigarrow r'$ are sharing-equivalent, well-behaved sets of wires, we conclude by the induction hypothesis. □

Lemma 71. *Sharing commutes with wiring: $D_\xi{\downarrow} = (D{\downarrow}_\xi){\downarrow}$.*

The set of wires ξ for $D{\downarrow}$ should of course be understood as the restriction of ξ to the sprouts of D which are still vertices of $D{\downarrow}$.

Proof. By induction on the size of ξ. If ξ is empty, the result is clear. Otherwise, let $\xi = \zeta \cup s \rightsquigarrow r$, where $s \rightsquigarrow r$ is maximal. By coherence of ξ, we can choose $s \rightsquigarrow r$ so that s is still a sprout of $D{\downarrow}$. By Lemma 66, r is mapped to a vertex $o(r)$ of D such that r and $o(r)$ generate sharing-equivalent subdrags. Now, $D_\xi = (D_{s\rightsquigarrow r})_\zeta$, and $D{\downarrow}_\xi = (D{\downarrow}_{s\rightsquigarrow o(r)})_\zeta$, and by the previous remark, $D_{s\rightsquigarrow r}$ and $D{\downarrow}_{s\rightsquigarrow r}$ are sharing-equivalent. We then conclude by the induction hypothesis. □

Lemma 72. *Sharing equivalence is closed under wiring.*

Proof. We are now given two sharing-equivalent drags D, D' and two sharing equivalent, well-behaved sets of wires ξ, ζ for D, D', respectively. Now,

$$\begin{aligned} D_\xi\!\downarrow &= (D\!\downarrow_\xi)\!\downarrow && \text{(by Lemma 71)} \\ &= (D'\!\downarrow_\xi)\!\downarrow && \text{(because } D \text{ and } D' \text{ are sharing equivalent)} \\ &= (D'\!\downarrow_\zeta)\!\downarrow && \text{(by Lemma 70)} \\ &= (D'_\zeta)\!\downarrow, \end{aligned}$$

showing that D_ξ and D'_ζ are sharing-equivalent. □

Using now Lemmas 68 and 72, we get:

Lemma 73. *Given two sharing-equivalent drags D, D', let $\langle E, \xi\rangle$ and $\langle E', \xi'\rangle$ be two sharing equivalent extensions of D and D', respectively. Then, $D \otimes_\xi E$ and $D' \otimes_{\xi'} E'$ are sharing-equivalent.*

The following important result summarizes the closure properties of sharing equivalence:

Theorem 74. *Sharing equivalence is closed under parallel and cyclic composition.*

10. Rewriting

Rewriting is often used as a method to decide congruences, or to describe syntactic transformations, the underlying congruence being implicit. The idea is that a congruence is an equivalence that is closed under composition with respect to extensions. This is the case for drags just like it is for terms, composition taking here the place of both context application and substitution.

As usual, rewriting is a precongruence. Symmetry is eschewed, so as to allow the unidirectional use of rewriting to decide whether two given drags are equivalent in the congruence generated by a given set of drag equations, thereby potentially reducing the nondeterminism involved in proof search.

10.1. **Rewrite Rules.** A rewrite rule serves to replace some drag pattern L by some other drag pattern R in a given drag D that contains L in a context defined by an extension $\langle E, \xi\rangle$ of L. That is, $D = E \otimes_\xi L$.

First, it is important to ensure that all roots and sprouts of L disappear in this composition, giving the notion of a rewriting extension:

Definition 75 (Rewriting extension)**.** *Given a drag L having no rootless isolated sprout, a* rewriting extension *of L is a extension $\langle C, \xi\rangle$ such that C is a linear drag which has no vertex nor variable in common with L, ξ_L is total and ξ_E surjective.*

Next, it is equally important to ensure that replacing L by R is possible, in other words that there exists an extension $\langle E', \xi'\rangle$ of R closely related to the extension of L, so that all roots and sprouts of R disappear in the composition $E' \otimes_{\xi'} R = D'$. This implies, in particular, that ξ' must be well-behaved and that each root in L must correspond to a root in R, hence suggesting a first proposal for a drag rewrite rule that generalizes the familiar notion of term rewrite rule:

Definition 76 (Patterns)**.** *A drag all of whose vertices are accessible is called a* right-pattern*. It is called a* left-pattern*, or simply* pattern*, if also all of its sprouts have predecessors.*

Definition 77 (Rewrite rules)**.** *A* drag rewrite rule $\eta : L \to R$*, written alternatively* $L \to_\eta R$*, has three components: a pattern* L*, a right-pattern* R*, and a multi-injective map* $\eta : \mathcal{R}(L) \to \mathcal{R}(R)$ *from the set of roots of* L *to the set of roots of* R*. A rule* $L \to R$ *is* stringent *if* $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$*.*

The case of term rewriting rules is quite simple: since L and R have a single root each, there is a unique possible map η. In [DJ19], L and R have ordered lists of roots of the same length, making η unique again. In both these cases, there is no need for η to be explicit. Having a multiset of roots forces us to specify η, whose rôle is to multi-injectively map the multiset of roots of L into the multiset of roots of R. Note that there may be strictly more roots in R than in L; having the same number would require η to be *multi-equijective*.

10.2. **Rewriting Relations.** We first consider "relational" rewriting. Given drags D, D', rewriting D to D' using rule $\eta : L \to R$ involves the following steps:

(1) match D against L: find a rewriting extension $\langle E, \xi \rangle$ of L such that $D = E \otimes_\xi L$;
(2) match D' against R: find a rewriting extension $\langle E', \xi' \rangle$ such that $D' = E' \otimes_{\xi'} R$;
(3) verify that the two extensions are compatible.

Compatibility means the following:

Definition 78 (Compatible extensions)**.** (1) *E and E' are equimorphic: $E \equiv_o E'$;*
(2) *for all sprouts s of E: $\xi'(o(s)) = \eta(\xi(s))$; and*
(3) *for all sprouts $t : x$ of L and $t' : x$ of R, the subdrags $E|_{\xi(t)}$ and $E'|_{\xi'(t')}$ are equimorphic.*

The rewriting switchboards map sprouts of E, E' to roots of L, R, respectively, and these mappings must fit with η, hence condition (2). Note that taking E and E' isomorphic would suffice: imposing that their sprouts are labeled by the same variables is possible because we distinguish both switchboards ξ for the left-hand side and ξ' for the right-hand side. Condition (3) expresses the property that the restrictions of ξ and ξ', to the sprouts of L and R, respectively, could be made into a single well-behaved set of wires, which will become important later.

We can now define *relational rewriting*:

Definition 79 (Rewriting)**.** *A drag D is in a* rewriting relation *with drag D'—using the rewrite rule $\eta : L \to R$ such that L and D have no vertex in common and η is a multi-injective map from the roots of L to the roots of R—if there exist two compatible rewriting extensions $\langle E, \xi \rangle$ of L and $\langle E', \xi' \rangle$ of R such that $D = E \otimes_\xi L$ and $D' = E' \otimes_{\xi'} R$.*

Since the extension drags E and E' must be equimorphic, it is tempting to take them to be identical. This is of course impossible in case D and D' do not share vertices, in which case only the sprouts of E and E' can be shared, which simplifies condition (2) already to $\xi'(s) = \eta(\xi(s))$. Usually, however, only D is given, and D' is defined by the rewriting process, in which case it is not necessary to generate new vertices for E'; we can take $E' = E$. In this case, we define rewriting as a computation mechanism.

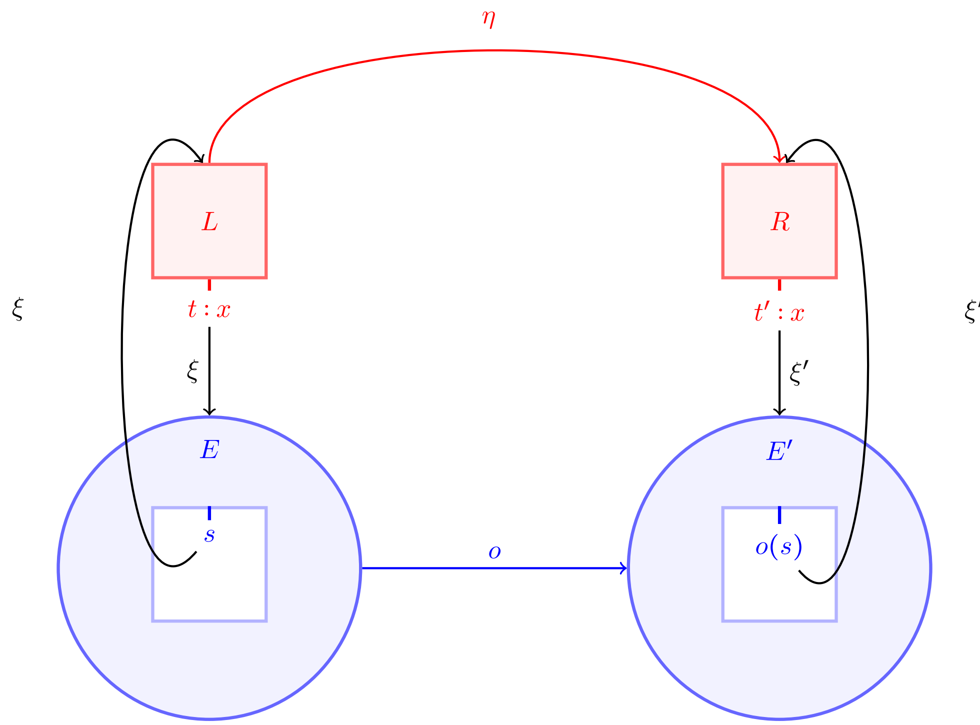


FIGURE 9. Compatible rewrite extension of a rewrite rule.

Definition 80 (Strong compatibility)**.** *Given a rule $\eta : L \to R$, two clean rewriting drag extensions $\langle E, \xi\rangle$ and $\langle E', \xi'\rangle$ of L and R, respectively, are* strongly compatible *if*

(1) *$E = E'$;*
(2) *for all sprout s of E, $\xi'(s) = \eta(\xi(s))$;*
(3) *for all sprouts $s : x$ of L and $t : x$ of R, the subdrags $E|_{\xi(s)}$ and $E|_{\xi'(t)}$ are equimorphic.*

Definition 81 (Functional rewriting)**.** *A drag D* rewrites *to a drag D' using the stringent rewrite rule $\eta : L \to R$ such that L and D have no vertex in common and η is a multi-injective map from the roots of L to the roots of R, if there exist two strongly compatible rewriting extensions $\langle E, \xi\rangle$ of L and $\langle E', \xi'\rangle$ of R such that $D = E \otimes_\xi L$ and $D' = E' \otimes_{\xi'} R$.*

We say that drag D rewrites *to drag D' with rewrite system $\mathcal{R}$, all of whose rules are stringent, denoted $D \longrightarrow_{\mathcal{R}} D'$, if D rewrites to D' using some rule $\eta \in \mathcal{R}$.*

Example 82. *Consider the two rewriting examples of Figure 10. The input drags to be rewritten and the resulting output drags are in black. The rule $h(x, x) \to k(x, x)$ is in red. Its various sprouts, all labeled x, are not shared; hence, we can use our naming conventions. The extension drags are in blue, and the switchboards in black.*

In the upper example, the right-hand side switchboard ξ' coincides roughly with the left-hand side switchboard ξ. Note that the left- and right-hand sides of the rule have a single root, the vertices h and k, respectively. Note also that these switchboards do satisfy strong compatibility.

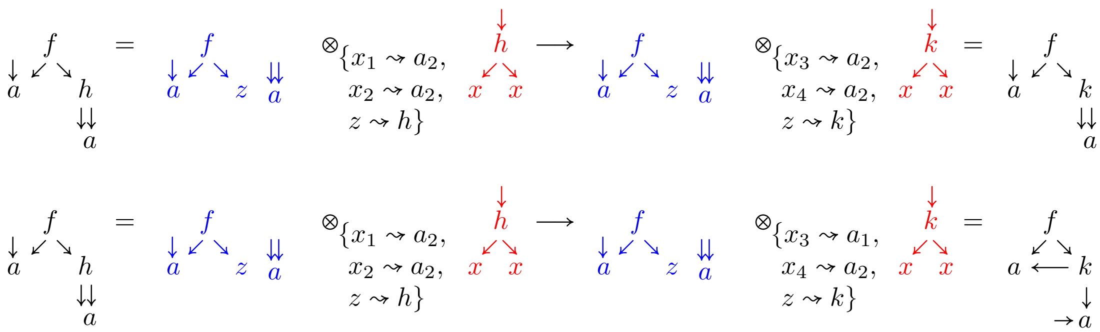


FIGURE 10. Rewriting example.

In the lower example, the switchboard ξ' differs from ξ in that both left-hand side x's are mapped to a_2 (there is no other choice) while the two right-hand sides x's are mapped to a_1 and a_2, respectively, which yields a quite different result. Note that a_1 and a_2 generate equimorphic subdrags reduced to a single vertex labeled a having two roots. Here, the right-hand side switchboard does not force sharing, that's why we can map x_3 and x_4 to different vertices.

Using the rule $h(x,x) \to k(x,x)$ in which x is shared in the left-hand side would yield exactly the same result with the same switchboards. In this example, the switchboard forces sharing on the left-hand side, even if the rule does not. □

So, the determination of a switchboard obeys precise rules, but leaves also some room for choosing the right-hand side switchboard among sometimes several possibilities. As a consequence, the result of a rewrite step is not entirely determined by matching, as is the case for terms. Note, however, that the two resulting drags obtained in Example 82 are very similar: they are the same up to sharing-equivalence.

Given now a rule $L \to R$ such that $\langle E, \xi\rangle$ and $\langle E, \zeta\rangle$ are two equimorphic extensions for R, the result of rewriting with that rule and these extensions will not depend upon which extension is used, up to sharing:

Theorem 83. *Let $D \longrightarrow_{L\to R} G$ and $D \longrightarrow_{L\to R} G'$, using compatible rewriting extensions. Then, G and G' are sharing-equivalent.*

Proof. Since compatible rewriting extensions are sharing-equivalent, R and R' are sharing-equivalent by Theorem 74. □

This result applies to rewriting as a computing device, but also to rewriting as a relation. Note that we have not required that variables of the right-hand side R of the rule $L \to R$ all occur in L. The choice of $\zeta(y)$, if y is such a variable, is given by matching G with respect to R, when using rewriting as a relation. When using rewriting as a computing device, every choice of $\zeta(y)$ will give a specific G, but any two strongly compatible choices of $\zeta(y)$ will give sharing-equivalent G's.

Example 84. *Figure 11 illustrates rewriting with a rule whose left-hand side originates from the example of wiring presented in Figure 7, and right-hand side is just a variable with 3 roots. Leftmost is the input drag, and rightmost is the result. Both are in black. The rule is written in red, the context in blue, the switchboard*

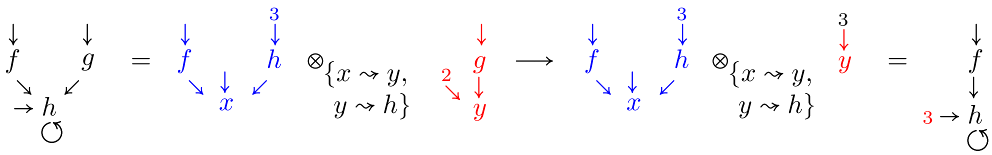


FIGURE 11. Rewriting with the red rule.

in black. Note that the number of roots of the resulting drag is one more than that of the original drag. The reader is invited to verify the result of the rewrite step by actually doing the composition calculation. □

A final question arises: Given a drag D, a rule $L \to R$, and a rewriting extension $\langle E, \xi \rangle$ for L, does there exist a rewriting extension $\langle E, \xi' \rangle$ for R? In general, yes, but there is a particular case for which this is not true. It may indeed be that the switchboard ξ, which is well-behaved for L, is not well-behaved for R. This happens in the following situation: u is a rooted internal vertex of L, $s : x$ is a sprout of L accessible from u, $s' : x$ is a rooted sprout of R such that $\eta(u) = s'$, $t : y$ is a sprout of E, $\xi(s) = t$, and $\xi(t) = u$. Switchboard ξ is well-behaved with respect to L because u is an internal vertex. Now, $\xi'(s') = t$ and $\xi'(t) = \eta(u) = s'$; hence, ξ' is not well-behaved. This is the only situation where this may arise, but this is why we always assumed the existence of one rewriting extension for L and one for R.

11. Drag Rewriting versus Term and Dag Rewriting

In this section, we consider term rewrite rules having possibly a root at their head, and (try to) apply them to terms that possibly involve sharing.

Consider a rule $f(x, x) \to g(x, x)$ with no roots on either side. The sprouts are x_1, x_2 in the left-hand side and x_3, x_4 in the right-hand side.

Let $t_1 = f(a, a)$ with vertices f (on top) and a_1 (shared). Let $t_2 = f(a, a)$, with vertices f, a_1, a_2 And let $t_1' = g(a, a)$, with the shared vertex a_1 being the same as above, and $t_2' = g(a, a)$ with vertices a_2 on the left and a_1 on the right.

- $t_1 \longrightarrow t_1'$ with extension drag E being the two-rooted vertex $a_1^{[2]}$ and switchboards being $\xi = \{x_1 \mapsto a_1, x_2 \mapsto a_1\}$ and $\xi' = \{x_3 \mapsto a_1, x_4 \mapsto a_1\}$.
- $t_2 \longrightarrow t_2'$ with extension drag $E = a_1^{[1]} \oplus a_2^{[1]}$, and switchboards $\xi = \{x_1 \mapsto a_1, x_2 \mapsto a_2\}$ and $\xi' = \{x_3 \mapsto a_2, x_4 \mapsto a_1\}$.
- $t_1 \longrightarrow t_2'$ with extension drags (two are needed here) $E = a_1^{[2]}$ and $E' = a_1^{[1]} \oplus a_2^{[1]}$ and switchboards $\xi = \{x_1 \mapsto a_1, x_2 \mapsto a_1\}$ and $\xi' = \{x_3 \mapsto a_2, x_4 \mapsto a_1\}$. Note that the vertex a_1 does not have the same number of roots in E and E'. Note also that E' cannot be identical to E since t_2' has additional vertices that do not originate from the rewrite rule, they must therefore come from the extension drag. This rewrite is therefore relational, using the clone a_2 of a_1.
- $t_2 \longrightarrow t_1'$. Take $E = a_1^{[1]} \oplus a_2^{[1]}$, $E' = a_1^{[2]}$, $\xi = \{x_1 \mapsto a_1, x_2 \mapsto a_2\}$ and $\xi' = \{x_3 \mapsto a_1, x_4 \mapsto a_1\}$. Note that the choice of E' allowed us to garbage collect the vertex $a_2^{[1]}$ of E implicitly. The other choice $E' = a_1^{[2]} \oplus a_2^{[1]}$

would yield as a result the expression $t'_1 \oplus a_2^{[1]}$, hence disabling garbage collection.

- $t_1 \longrightarrow a_3^{[1]} \oplus t'_1$, where a_3 is a fresh copy of a, that is, a *clone* of a. Take $E = a_1^{[1]} \oplus a_2^{[1]}$, $E' = a_1^{[2]} \oplus a_3^{[1]}$, with ξ and ξ' as above. Here, we have achieved cloning and garbage collection at the same time.

Terminating computations, to which we are partial, are incompatible with cloning. In general, functional rewriting restricts context extensions so as to avoid cloning and allow one to attain termination. Relational rewriting, on the other hand, is more lax regarding cloning and nontermination.

We provide now more formal statements comparing term and dag rewriting to drag rewriting. To this end, we encode a term t as a drag $[\![t]\!]$ by adding a root at its head. This applies of course to rules, so that a set of term rewrite rules R may be denoted by $[\![R]\!]$ when encoded as drag rewrite rules. Assuming that left-hand sides of term rewriting rules are not variables (hence are patterns), we have:

Lemma 85. *Let R be a set of term rewriting rules and s, t two terms. Then, $s \longrightarrow_R t$ iff $[\![s]\!] \longrightarrow_{[\![R]\!]} [\![t]\!]$.*

The proof of this requires a proof of the one-step case with some rule $l \to r$, which itself follows from the definitions of term and drag rewriting, from Lemma 87 (relating term matching using l with drag matching using $[\![l]\!]$), and from Lemma 88 (relating drag matching using $[\![r]\!]$ and term matching using r).

Note that this result is valid for the relational drag rewriting model only. It does also hold for the drag rewriting model when the term rewriting rules are linear.

Considering now the case of dags, sharing requires a different encoding, since there may be arbitrarily many edges going from the context dag to arbitrary internal vertices of the rewritten subdrag. This time, we won't add roots to the drags that need be rewritten, but will instead add arbitrarily many roots (possibly none) at each vertex—including sprouts—of the left-hand side drag of the rule $d \to e$. For each such encoding of d, an equal number of roots should be added to e, so as to be able to define the map η. Using the same notation for encoding rules, we arrive at the following:

Lemma 86. *Let R be a set of dag rewriting rules and s, t two dags. Then, $s \longrightarrow_R t$ iff $s \longrightarrow_{[\![R]\!]} t$.*

The relational rewriting model is therefore quite powerful, strictly more powerful than the functional model, and strictly more powerful as well than the term or dag rewriting models for rewriting terms or dags since by switching from a drag extension E to a different one E', it allows a fine tune up of garbage collection and cloning, which the other models do not permit. The functional rewriting model, on the other hand does not enjoy this extra power, and that is why term and dag rewriting, which are likewise functional, also do not.

12. Categorical Interpretation of Drag Rewriting

Our definition of drag rewriting is concrete: drags are specific concrete (multi-) graphs, and drag rewrite rules are pairs of drags whose roots are injectively related, used to rewrite other drags. We found out, however, that drags and their morphisms form a category. What categorical constructions correspond to rewriting drags in that category? That's the question we answer in this section.

We start with the operation of matching a drag D against a drag L. The existence of a monomorphism from L to D is the traditional definition of matching used in categorical approaches, such as DPO. The existence of an extension $\langle C, \xi \rangle$ such that $D = L \otimes_\xi C$ is our rewriting definition matching. We investigate their relationship below.

Lemma 87. *Given two disjoint drags L, C and a switchboard ξ for L, C, the natural injection from L to $C \otimes_\xi L$ is a monomorphism.*

Proof. By Lemma 26, the natural injection from L to $L \oplus C$ is a monomorphism. By Lemma 40, the natural injection from $L \oplus C$ to $L \otimes_\xi C$ is a monomorphism. We conclude by Lemma 21. □

Next, we consider the converse, namely, the existence of an extension when given the injection.

Lemma 88. *Given a drag D, a drag L with no rootless isolated sprout and no sprout in common with D, and an injection $o : L \hookrightarrow D$, there exists a rewriting extension $\langle C, \xi \rangle$ of L such that $D = C \otimes_\xi L$.*

Proof. Without loss of generality, sprouts of D, if any, will be considered internal vertices of D, since they will not belong to the domain of ξ. In what follows, we successively (a) construct the context extension C, (b) then the switchboard ξ, and finally (c) verify that putting them together we have $D = C \otimes_\xi L$.
(a). Construction of context C:

- Labeled internal vertices. Let V be the vertices of D and I the internal vertices of L (which are also internal vertices of D by the assumption that o is an injection). Then, the set of internal vertices of C will be $W = V \setminus I$, each one equipped with its label in D. Edges of C between vertices of W are just those of D.
- Labeled sprouts. The set S of sprouts of C consists of fresh sprouts $t_{u,i} : x_{u,i}$, bijectively associated with the pair (u, i) for each entering edge $u \longrightarrow^i v$ in D—with $u \in W$ and $v \in I$, plus sprouts $t_{u,i} : y_{u,i}$ for each creating edge $u \longrightarrow^i v$ in D, such that $u, v \in I$, but $u \longrightarrow^i v$ is not an edge of L, $u \longrightarrow^i s$ is an edge in L and $o(s) = v$. Notice that the edge $u \longrightarrow^i s$, with $o(s) = v$, must exist by the definition of premorphism. Notice also that, by definition, the variables of any two sprouts in C must be different, implying that the context C will be linear.
- Edges. The set of edges of C consists of all edges $u \longrightarrow^i v$ in D, such that $u, v \in W$, plus edges $u \longrightarrow^i t_{u,i}$, for each entering edge $u \longrightarrow^i v$ in D with $u \in W$ and $v \in I$. These sprouts are not isolated.
- Roots. Finally, each vertex v in W is equipped with roots so that v has the same indegree in C and D. Sprouts $t_{u,i}$, $t_{u,i} : y_{u,i} \in C$, associated to the creating edge $u \longrightarrow^i v$ in D, are equipped with a single root, hence will be rooted, isolated sprouts.

(b). Construction of switchboard ξ:

- For each creating edge $u \longrightarrow^i s$ of L, $\xi_L(s) = t_{u,i} : y_{u,i}$. For any other sprout $s \in L$, $\xi_L(s) = o(s)$.
- For each sprout $t_{u,i} : x_{u,i} \in C$, associated to entering edge $u \longrightarrow^i v$ in D, $\xi_C(t_{u,i} : x_{u,i}) = v$, and for each sprout $t_{u,i} : y_{u,i} \in C$, associated to creating edge $u \longrightarrow^i v$ in D, $\xi_C(t_{u,i} : x_{u,i}) = v$.

We are left with showing that ξ is a switchboard. The union $\xi_L \cup \xi_C$ is coherent and well-behaved: It is coherent since o is a premorphism and C is linear; by definition $\xi_L \cup \xi_C$ is functional and well-founded. Finally, $\xi_L \cup \xi_C$ is injective, since o is a morphism and, hence, root preserving. Therefore, $\langle C, \xi \rangle$ is a rewriting extension.
(c). Verification that $D = C \otimes_\xi L$: Internal vertices of D and $C \otimes_\xi L$ coincide, and so too their labeling. We show that edges also coincide by inspecting all categories of edges $u \longrightarrow^i v$ of D:

- $u, v \in W$. By construction of C, $u \longrightarrow^i v$ is an edge of C and therefore of $C \otimes_\xi L$.
- $u, v \in I$ and $u \longrightarrow^i v$ is an edge of L. Then it is an edge in $C \otimes_\xi L$.
- $u, v \in I$, but $u \longrightarrow^i v$ is not an edge of L; hence it is a created edge of D corresponding to the creating edge $u \longrightarrow^i s$ of L. By construction, $\xi_L(s) = t_{u,i}$ and $\xi_C(t_{u,i}) = v$; hence $u \longrightarrow^i v$ is an edge in $C \otimes_\xi L$. This case shows the need for bouncing from L to C and back from C to L.
- $u \in I$, $v \in W$. By the definition of an injection, the ith edge issuing from u in L must be of the form $u \longrightarrow^i s$, where s is a sprout such that $o(s) = v$. By construction, $\xi_L(s) = v$; hence $u \longrightarrow^i v$ is an edge of $C \otimes_\xi L$.
- $u \in W$, $v \in I$. Hence $u \longrightarrow^i v$ is an entering edge. By construction, C includes a sprout $t_{u,i} : x_{u,i}$ and an edge $u \longrightarrow^i t_{u,i}$, with $\xi_C(t_{u,i}) = v$; hence $u \longrightarrow^i v$ is an edge in $C \otimes_\xi L$.
- Since all sprouts $t_{u,i}$ added to C are bi-univocally associated with an edge $u \longrightarrow^i v$ of D, either creating or entering, there are no other edges in $C \otimes_\xi L$.

In case L is a rooted isolated sprout, the constructed context C is identical to D, and the switchboard ξ contains the single wire $s \rightsquigarrow o(s)$. Verification then succeeds trivially since isolated sprouts are identities for product in this case by Lemma 60. ☐

Example 89. *Let D be a drag with two internal vertices labeled $h^{[1]}$ and $f^{[1]}$, both of arity 3, and edges $h \longrightarrow^1 f$, $h \longrightarrow^2 f$, $h \longrightarrow^3 h$, and $f \longrightarrow^1 h$, $f \longrightarrow^2 f$, $f \longrightarrow^3 h$. Now consider the drag $L = f^{[4]}(x_1^{[1]}, x_2^{[1]}, x_3)$ with the injection o mapping its four vertices f, x_1, x_2, x_3 to f, h, f, h, respectively, and $o_R(x_1^{[1]}) = h \longrightarrow^3 h$, $o_R(f^{[3]}) = \{h \longrightarrow^1 f, h \longrightarrow^2 f, f \longrightarrow^3 f\}$.*

Let now C be the drag $h^{[4]}(y_1, \textsc{self}, y_3) \oplus z^{[2]}$. We now construct the expected extension by processing all missing edges in turn:

1. *$f \longrightarrow^2 f$: add $z^{[2]}$ to C, define $\xi_L(x_2) = z$ and $\xi_C(z) = f$; remove a root from f and $f \longrightarrow^2 f$ from $o_R(f)$;*
2. *$f \longrightarrow^1 h$: define $\xi_L(x_1) = h$ and remove $f \longrightarrow^1 h$ from $o_R(h)$;*
3. *$f \longrightarrow^3 h$: define $\xi_L(x_3) = h$ and remove $f \longrightarrow^3 h$ from $o_R(h)$;*
4. *$h \longrightarrow^1 f$: define $\xi_C(y_1) = f$ and remove $h \longrightarrow^1 f$ from $o_R(f)$;*
5. *$h \longrightarrow^2 f$: define $\xi_C(y_2) = f$ and remove $h \longrightarrow^2 f$ from $o_R(f)$.*

We therefore obtain the extension:

$$(h^{[3]}(y_1, y_2, \textsc{self}) \oplus z^{[2]}, \ \{x_1 \rightsquigarrow h, x_2 \rightsquigarrow z, z \rightsquigarrow f, x_3 \rightsquigarrow h, y_1 \rightsquigarrow f, y_2 \rightsquigarrow f, y_3 \rightsquigarrow x_1, z \rightsquigarrow h\})$$

We observe that mapping x_2 to z and then z to f produces the edge $f \longrightarrow f$, while other edges are produced more directly, as pointed out above, not being created edges of L in D. ☐

The previous lemmas imply that matching based on composition and matching based on the existence of an injection coincide:

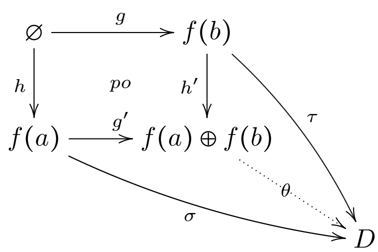


Figure 12. Pushout for terms as drags.

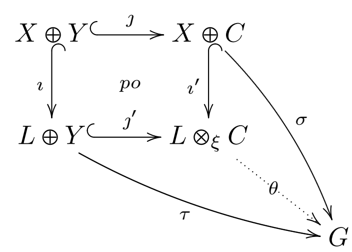


Figure 13. Pushout for drags.

Theorem 90. *Given a drag D, a drag L with no rootless isolated sprout and no sprout in common with D, there exists an injection $o : L \hookrightarrow D$ iff there exists a rewriting extension $\langle C, \xi\rangle$ of L such that $D = C \otimes_\xi L$.*

We have not claimed uniqueness of the rewriting extension $\langle C, \xi\rangle$ when given D, L, and ι, and there are indeed many rewriting extensions satisfying Lemma 88, hence Theorem 90. The relevance of uniqueness lies in the fact that given D, L, o, the result of rewriting D with $L \to R$ at o would then be deterministic, as is usually expected from a functional rewriting mechanism. Uniqueness could indeed be achieved by introducing the definition of a *reduced extension*, imposing two requirements: The switchboard ξ should not bounce between L and C more than necessary. And isolated sprouts of the extension context C should have a single root, hence cannot be targets for two different wires of the switchboard. This question merits further investigation, and it would be interesting to have both a matching and an unification algorithm for this version of drags, in the style of [JO23].

We now consider rewriting a drag D to a drag D' using a rule $L \to R$, and show that D and D' are certain pushouts in the category of drags. We first recall that the category of *terms* fails to have pushouts in general.

Example 91. *Let $f(a)$ and $f(b)$ be terms, where a and b are distinct constant symbols in the signature, the vertices being named f_1, a, f_2, b, respectively. Note that f_1 and f_2 must be different by our assumption that sharing must propagate to all successors. Let now h and g be two morphisms mapping the empty term $\varnothing$ to $f(a)$ and $f(b)$, respectively. Then, there is no pushout of h and g in the category of terms.*

There is one, however, in the category of forests, and the same pushout holds indeed in the category of drags, obtained by mapping $f(a)$ and $f(b)$ to $f(a) \oplus f(b)$, as shown in Figure 12. Since a and b are distinct vertices, D must contain two subdrags $f(a)$ and $f(b)$, which must be distinct by our assumption that sharing propagates to successors, images of $f(a)$ and $f(b)$ by the morphisms σ and τ, respectively. Then the requirement that $\theta \circ g' = \sigma$ and $\theta \circ h' = \tau$ implies that θ must map $f(a)$ and $f(b)$ to those very same subterms $f(a)$ and $f(b)$ of D, implying its uniqueness. Note that it is important to start from the empty term, so that there is no requirement on the morphisms σ and τ. □

Regarding pushouts, the category of drags seems better behaved than that of terms. There are in fact enough pushouts for our needs:

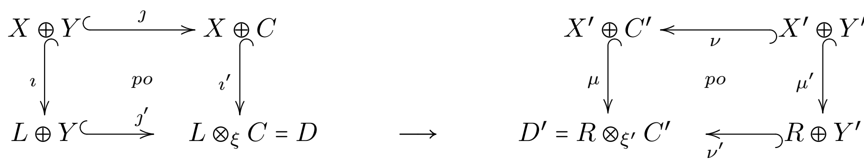


FIGURE 14. Double pushout diagram for drag rewriting. Here (C, ξ) and (C', ξ') are compatible rewrite extensions; X and Y are the sets of sprouts of L and R, respectively; and Y and Y' are the sets of sprouts of C and C', respectively.

Lemma 92. *Given disjoint drags L and C, switchboard ξ for (L, C), and sprouts X, Y of L and C, respectively, let $\imath, \jmath$ be the identity monomomorphisms from $X \oplus Y$ to $L \oplus Y$ and $C \oplus X$, respectively. Then, there exist monomorphisms $\imath'$ and $\jmath'$ from $L \oplus Y$ and $X \oplus C$ to $D = L \otimes_\xi C$, respectively, which form a pushout for $(\imath, \jmath)$.*

Note that Figure 12 is a particular case of Figure 13 in case the drags L and C are ground terms, implying that $X \oplus Y = \varnothing$. Finally, the property generalizes to the case of compatible drags L, C by taking the sprouts of $L \oplus C$ instead of $X \oplus Y$. (Shared sprouts should not be repeated; the reader can adapt $L \oplus Y$ and $X \oplus C$ accordingly.) The property generalizes even more by replacing $X \oplus Y$ by a sum of three components, a subdrag B of L, the sprouts of $L \smallsetminus B$, and the sprouts Y of C. The reader is again invited to adapt the rest of the pushout diagram.

Proof. Let $\imath'$ be the identity map for the internal vertices of C, equal to ξ_C for the sprouts Y of C and to ξ_L for the sprouts X of L. It is easy to see that $\imath'$ is a monomorphism, since indegree preservation is a property of cyclic composition. The same holds of course for $\jmath'$.

Let now G be a drag and $\sigma : X \oplus C \to G$ and $\tau : L \oplus Y \to G$ be morphisms such that $\tau \circ \imath = \sigma \circ \jmath$. Since ι' is injective on internal vertices of C, we define θ on internal vertices of C as $\sigma \circ (\imath')^{-1}$. Similarly, for the internal vertices of L, $\theta = \tau \circ (\jmath')^{-1}$, showing that θ is the unique map such that $\sigma = \theta \circ (\imath')^{-1}$ on internal vertices of C and $\tau = \theta \circ (\jmath')^{-1}$ on internal vertices of L. Without loss of generality, we can assume that G and $L \otimes_\xi C$ are ground; hence θ is defined on all vertices of $L \otimes_\xi C$. Furthermore, it is clear that θ is a morphism.

We are left with showing that $\sigma = \theta \circ (\imath')^{-1}$ and $\tau = \theta \circ (\jmath')^{-1}$ on $X \cup Y$, which follows from the assumption that $\tau \circ \imath = \sigma \circ \jmath$ and from the definitions of $\imath'$ and $\jmath'$. □

Given now a rewrite $D = L \otimes_\xi C \longrightarrow R \otimes_{\xi'} C' = D'$ such that (C, ξ) and (C', ξ') are compatible rewrite extensions, we can apply Lemma 92 to both products, and get the double pushout diagram presented in Figure 14. Drag rewriting can therefore be understood in terms of two pushout constructions strongly related by compatible rewriting extensions.

13. DISCUSSION

In the course of this study of graph rewriting, we have made a number of choices among alternatives, motivated by what seemed to us either more general and useful, or simpler and more convenient.

In Remark 6, we explained why we have chosen to consider multisets of roots, rather than lists as in [DJ19] or sets. This design choice impacted our definition of composition, for which we decided to maintain a strong invariant: the indegree of each individual vertex. We indeed tried an alternative, using root transfer as in [DJ19], which resulted in more complex technicalities that we were not able to resolve in a satisfactory manner. Root transfer considers roots as plugs waiting for a connection *there*, while indegree preservation considers roots as wires waiting for a connection *at the other end*.

Another issue is how to understand multiple instances of variables. We have already explained in Section 6.3 why we require equimorphism of the subdrags connected to different sprouts with the same label, rather that the weaker isomorphism suggested in [DJ19] or the stronger identity relation used in [DJ19]. We have also seen that this gives us the very helpful Lemmas 21 and 40. The latter is extremely important in that it allowed us to relate, in Section 12, two completely different notions of matching: the traditional one, matching as a monomorphism, and the new one, matching as a drag extension. This relationship is a major justification for the drag model. But is equimorphism the best possible answer?

In the remainder of this section, we hint at variations that may extend the capabilities of the drag model. First, we briefly discuss a more general definition of coherence of a set of wires. Second, and significantly, we consider how sharing can be improved by a definition of rules allowing their left- and right-hand sides to share subdrags. Next, we suggest dropping the fixed arity of labeled vertices, and consequently hint at a more flexible graph-rewriting model for which sprouts and roots are particular cases of a more general notion of connector.

13.1. **Coherent Sets of Wires.** There is a slight potential for generalization here, replacing *equimorphism* by *sharing equivalence* in the definition of a coherent set of wires. This variant would require changes in the definition of monomorphisms so as to preserve Lemma 40, a change that should not impact Theorem 23 since isomorphisms preserve sharing equivalence. The computation of a product $L \otimes_\xi C$ can render two subterms of L equimorphic, or even sharing equivalent, and they might then be shared in the resulting drag. As a consequence, matching would no longer be injective on internal vertices. Note that part of the DPO community uses non-injective matching, although in [HMP98] it is shown that injective matching is more powerful. We won't explore that path here, but it is worth mentioning as a potential area of future investigation.

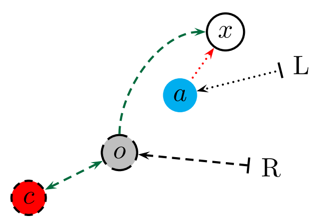
13.2. **Rule Sharing.** Because the left-hand side and right-hand side of a rewrite rule are separate drags, they do not share any subterm, not even a sprout. This has a drawback, in case they have identical subexpressions that could be shared: the identical subexpression will be eliminated before to be reconstructed, a waste of space and time. Furthermore, different sprouts labeled by a same variable can be mapped to different vertices generating equimorphic drags, hence implying some copying operations. To force sharing, we will move from rules as pairs of drags to rules as a single drag equipped with pairs of lists of roots, one for the left-hand side, and one for the right-hand side.

Definition 93 (Rules as a single drag)**.** *A* drag rewrite rule *is a drag D equipped with two submultisets, L and R, of $\mathcal{R}(D)$, such that $\mathcal{R}(D) = L \cup R$, and a multi-injective map η from $\mathcal{R}(L)$ to $\mathcal{R}(R)$. Overloading notation, we refer to the two* *subdrags* $D|_L$ *and* $D|_R$ *by their (sets of) roots,* $L = D|_L$ *and* $R = D|_R$*. We assume that* L *is a pattern and* R *a right-pattern, and write* $L \to_\eta R$ *for the rule. A rule is* stringent *if* $\mathcal{V}ar(D|_R) \subseteq \mathcal{V}ar(D|_L)$.

Being a single drag, the left-hand and right-hand sides of a rule can easily share subdrags, which will not need be eliminated and reconstructed when rewriting. Note that even if root vertices can be shared, the roots need not be equally distributed among L and R. For example, a root vertex $r^{[4]}$ can be shared by L and R, with one root belonging to L and three to R. More generally, if r is a root with multiplicity n, then r can be a root of multiplicity p in L, and a root of multiplicity $n-p$ in R. Any multi-injective map η would then do, the p roots of L being possibly all mapped to different vertices by the switchboard, making this notion of shared rule quite flexible.

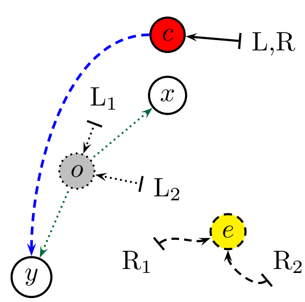
In Section 4, we saw an example of drag rewrite rules, in two versions. The two drags on the two sides of rules either shared nothing but variables in common, or else they potentially also shared internal vertices.

In the fully shared subdrag scenario, the left- and right-hand sides are *both* parts of the same drag. For the first rule of the example of Section 4, we now have the following combined drag:



We are using dotted arrows for edges that are in the left side of the rule only, and dashed arrows when it's only on the right; were an edge in both, we'd leave it solid (like some of the roots in the second rule below). Left roots are noted by an L; right ones by R. Internal vertices are only on one side are likewise dotted or dashed. So, this rule adds gray o and red c for each blue a, erasing the latter.

The second rule looks now like this:



The red c vertex is shared; a yellow e is introduced in place of the deleted gray o, with incoming edges L_1 and L_2 redirected to roots R_1 and R_2, respectively.

Example 94. *Consider the rule described by the drag* $f_1^{[1]}(f_2^{[1]}(x))$ *with two roots, the left-hand side root pointing at the upper occurrence of* f *(*$L = f_1$*), and the* *right-hand side root pointing at the inner occurrence of* f *(*$R = f_2$*). Rewriting with this rule amounts to eliminating an* f*, the outermost one, from any drag* D *having two consecutive symbols* f*, for example* $D = f(f(a))$*. The entire drag* $f(a)$ *which results from the computation is therefore the very subterm* $f(a)$ *of* $f(f(a))$*, which goes beyond what is usually done in term rewriting implementations, where only* a *would derive from the original term* $f(f(a))$*.*

Consider now the rule given by the drag $f_1^{[1]}(f_2(x)) \oplus f_3^{[1]}(x)$ *made of the two terms* $f(f(x))$ *and* $f(x)$ *with no common subexpression, and two roots,* $L = f_1$ *and* $R = f_3$*. Rewriting the term* $f(f(a))$ *with that rule has a completely different effect: It still eliminates the topmost* f*, but it will now generate a new vertex labeled* f*, and possibly (but not necessarily, depending whether the two sprouts labeled* x *are shared or not) a new vertex labeled* a*, resulting in a term* $f(a)$ *that may be an entire, or only partial, copy of the subterm* $f(a)$ *of the term* $f(f(a))$*.* □

13.3. **Varyadic Labels.** Drags were designed for generalizing terms and term rewriting. Accordingly, a vertex of a drag comes with a label equipped with a fixed arity that governs the number of successors of that vertex, a constraint that has not, however, been central to the theory of drags developed here. Extending the model to deal with arbitrary graphs is possible by relaxing the fixed arity constraint, allowing for bounded or even unbounded arities.

In the fixed-arity model, it is crucial that wiring does not change the number of outgoing edges at a vertex. It follows that only sprouts can be mapped to other vertices, implying that decomposing a drag into smaller pieces can only be done by cutting its edges.

This limitation disappears with varyadic arities. Thus an alternative would be to decompose drags by "slicing" apart vertices instead of cutting edges. The incident edges (regardless of direction) of the vertex are then split between the slices. Dissecting an internal vertex creates a new "snap", along with the leftover "base" vertex. Composition (or wiring) connects then snaps with vertices, which may also be snaps. Decomposition of a drag into two is straightforward in this alternate model, as would be decomposition into single-edge atoms.

14. Related Work

There are several competing approaches to graph rewriting. We list some of the more closely related ones.

14.1. **DPO.** Introduced in the early seventies, the double-pushout (DPO) approach [EPS73] is the best studied and most popular approach to graph transformation. There are many varieties of graphs that may be of interest in different contexts. For example, one may work with directed or undirected graphs; they may be typed or untyped; they may be labeled, unlabeled, or include attributes that represent values stored in vertices or edges; they may be graphical structures like Petri nets or state-transition diagrams, or they even may be drags. It should be clear that studying graph rewriting separately for each kind of graph is a waste of time since there is almost no difference between rewriting a directed or an undirected graph or any other kind of graphical structure. Using categorical constructions allows the DPO approach to describe and study at one and the same time rewriting for all manner of structures that satisfy some given properties. The DPO approach is currently defined for any category of objects that is *adhesive* [EEPT06, LS06] (or $\mathcal{M}$-*adhesive* [EGH+14, EGH+12]). This includes most graph categories as well as other graphical structures, and other categories of objects, like sets, bags, or algebraic specifications.

In Section 12, we showed that the DPO construction applies to drags as well, and can be extended so as to be relational rather than functional. The definition of a rule $L \to R$ as a single drag with left- and right-hand sides roots L and R as introduced at Definition 93 looks very much like a DPO rule, the subgraphs accessible from both L and R serving as the interface. The only difference is that we do not force the rewriting extensions of L and R to be identical, but rather to be compatible, which makes sense for concrete graphs. A direct benefit of compatibility combined with a relational definition of rewriting is that this extended version of DPO has a built-in ability for erasing and cloning subdrags. Furthermore, drag rules can be nonlinear, on the left or right, while DPO rules are essentially linear. We can therefore claim to have solved the problem of allowing nonlinear variables in drag rewrite rules, a problem the importance of which was stressed already in [PEM86], and which has remained unsolved since then, despite numerous attempts. Non-linearity is permitted here by a subtle definition of morphisms for drags with variables. We believe that similar ideas should scale to graphs whose vertices can have varyadic labels, along the lines suggested in Section 13.3. Making sense of nonlinear rules (and morphisms) in the case of abstract graph categories might require additional nontrivial properties of morphisms yet to be elaborated.

14.2. **Algebraic Approaches.** Several other algebraic approaches, like Agree [CDE+15], PBPO [CDE+19], and PBPO+ [OER21], have been defined to overcome some limitations of the DPO approach, such as the ability to erase or clone nodes. For us, cloning and erasing can be implemented by drag rewriting.

14.3. **Patches.** A framework similar to ours has been recently developed by Overbeek and Endrullis [OE20], but which is designed for graphs whose vertices have variable arity. For composition, they employ "patches"—a device similar to our switchboard, which adds connecting edges between the two components. Likewise, they have an analogue to roots cum sprouts (rather like the snaps suggested at Section 13.3), which allows one to constrain the permitted shapes of subgraphs around a match for a left-hand side, and also to specify how the subgraphs should be transformed. Transformations include rearrangement, deletion, and duplication of edges. PBPO+ [OER21], which was developed in a categorical framework, can be seen as a conceptual successor to patches and has been proposed as a unifying notion.

14.4. **Graphs with Interfaces.** The idea of building graphs using some kind of composition operation, by gluing some selected nodes of the graphs involved, which are considered interfaces, is already quite old, going back to the work of Bauderon and Courcelle [BC87]. In the context of DPO, graphs with interfaces and their transformation have been studied by Bonchi, Corradini, Gadducci, et al.; see, for instance, [CG99, Gad07, BGK+17]. The main difference is that they only consider sequential composition; they don't consider the possibility that sprouts of one drag are connected to roots of another and vice versa. In our terms, this means that in this case composition is limited to *one-way switchboards* [DJ19].

14.5. **String Diagrams.** String diagrams are a restricted graphical syntax for representing computational models used in various fields, including programming language semantics, circuit theory, and control theory. Mathematically, string diagrams are the terms of symmetric monoidal theories, which generalize algebraic theories in a way that makes them suitable for expressing resource-sensitive systems in which variables cannot be copied or discarded at will. String diagrams enjoy a restricted composition operator, in the sense that it is based on one-way switchboards, as introduced in [DJ19]. Rewriting of string diagrams is defined as a specific instance of DPO rewriting with interfaces (DPOI), called convex rewriting, for a category of labeled hypergraphs that correspond to string diagrams [BGK+22a, BGK+22b].

15. Conclusion

The drag framework was conceived so as to apply to a specific category of graphs, namely drags, and to generalize the standard term rewriting and dag models to drags. As a consequence, drags are graphs equipped with specific vertices, called sprouts, labeled with variables, while the other, internal vertices are labeled by function symbols equipped with an arity that specifies the number of their outgoing edges. In addition, vertices are equipped with roots that provide them with the potential for creating new edges.

The major originality of the drag model is to base the matching of a given drag D with respect to a left-hand side of rule L on the existence of a pair made of a context drag C and a switchboard ξ so that D is the product of L and C with respect to ξ. In this view, the switchboard ξ maps a sprout s of each drag to a rooted vertex r of the other drag, provided r has at least as many roots as the number of incoming edges and roots of s. Computing the product amounts to redirecting to r all edges incoming to s and removing from r an equal number of roots, an operation that leaves the indegree of r unchanged. Rewriting amounts then to replacing L by R, that is, to computing the new drag resulting from the product of the context C with the right-hand side R with respect to the switchboard ξ. This assumes the existence of an injective mapping from the roots of L to the roots of R.

We have indeed succeeded, inasmuch as our new drag model appears to generalize the term and dag rewriting models very smoothly, something that our former drag model [DJ19] could not do. Furthermore, it even generalizes the term and dag rewriting models when applied to terms and dags by having two new built-in capabilities: sharing and cloning. By this we mean that we are able to specify *formally* at each rewrite step which subdrags should be shared and which should be duplicated.

The most widely accepted and most widely used graph-rewriting model is DPO. While DPO was conceived so as to apply to various categories of graph structures, namely the adhesive categories, its expressivity is limited by the absence of variables, one consequence of which is the infeasibility of cloning.

A natural question then follows: Can graphs be equipped with variables, and can these variables be used within the DPO model? This question was actually raised long ago [PEM86], but to date no satisfactory answer has been proffered [Hof05], despite several attempts, most notably that of [HP96]. We give here a general answer to that question for the category of drags, thanks to the drag's notion of a variable being a one-way channel, to the notion of switchboard, which allows one to compose graphs in a very general way, and to a notion of morphism (and monomorphism) for non-ground drags. Matching a left-hand side of rule can then be defined either via composition or via the existence of a monomorphism in the obtained category, while rewriting can be defined by replacement or by a double pushout. Even more, a slight generalization of DPO is suggested that inherits the cloning and sharing capabilities of composition based drag rewriting.

Even more interesting are the following related questions: Can composition be defined for arbitrary graphical structures, or—more precisely—for arbitrary objects belonging to some adhesive category? Is adhesivity required for that purpose? Are variables needed for that purpose?

Future work on our part will be devoted to answering these questions, or at least some of them.

References

[BC87] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Math. Syst. Theory*, 20(2-3):83–127, 1987. doi:10.1007/BF01692060.

[BGK+17] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Confluence of graph rewriting with interfaces. In Hongseok Yang, editor, *Programming Languages and Systems – 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 141–169. Springer, 2017. doi:10.1007/978-3-662-54434-1_6.

[BGK+22a] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: Rewriting with Frobenius structure. *Journal of the ACM*, 69(2):14:1–14:58, 2022. doi:10.1145/3502719.

[BGK+22b] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory II: Rewriting with symmetric monoidal structure. *Math. Struct. Comput. Sci.*, 32(4):511–541, 2022. doi:10.1017/S0960129522000317.

[CDE+15] Andrea Corradini, Dominique Duval, Rachid Echahed, Frédéric Prost, and Leila Ribeiro. AGREE - algebraic graph rewriting with controlled embedding. 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 35–51. Springer, 2015. doi:10.1007/978-3-319-21145-9_3.

[CDE+19] Andrea Corradini, Dominique Duval, Rachid Echahed, Frédéric Prost, and Leila Ribeiro. The PBPO graph transformation approach. *J. Log. Algebraic Methods Program.*, 103:213–231, 2019. doi:10.1016/j.jlamp.2018.12.003.

[CG99] Andrea Corradini and Fabio Gadducci. An algebraic presentation of term graphs, via gs-monoidal categories. *Appl. Categorical Struct.*, 7(4):299–331, 1999. doi:10.1023/A:1008647417502.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier, 1990.

[Cou93] Bruno Courcelle. Graph rewriting: A bibliographical guide. In Hubert Comon and Jean-Pierre Jouannaud, editors, *Term Rewriting, French Spring School of Theoretical Computer Science, Font Romeux, France, May 17–21, 1993, Advanced Course*, volume 909 of *Lecture Notes in Computer Science*. Springer, 1993. URL: https://www.labri.fr/perso/courcell/Textes/BiblioReecritureGraphes(1995).pdf, doi:10.1007/3-540-59340-3_6.

[CS18] Horatiu Cirstea and David Sabel, editors. *Proceedings Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2017, Oxford, UK, September 2017*, volume 265 of *EPTCS*, 2018. URL: http://arxiv.org/abs/1802.05862.

[DJ18] Nachum Dershowitz and Jean-Pierre Jouannaud. Graph path orderings. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-22)*, volume 57 of *EPiC Series in Computing*, pages 307–325. EasyChair, 2018. URL: `https://easychair.org/publications/paper/8DzT`, `doi:10.29007/6hkk`.

[DJ19] Nachum Dershowitz and Jean-Pierre Jouannaud. Drags: A compositional algebraic framework for graph rewriting. *Theoretical Computer Science*, 777:204–231, 2019. `doi:10.1016/j.tcs.2019.01.029`.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[EGH+12] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. $\mathscr{M}$-adhesive transformation systems with nested application conditions. Part 2: Embedding, critical pairs and local confluence. *Fundam. Inform.*, 118(1–2):35–63, 2012. `doi:10.3233/FI-2012-705`.

[EGH+14] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. $\mathscr{M}$-adhesive transformation systems with nested application conditions. Part 1: Parallelism, concurrency and amalgamation. *Math. Struct. Comput. Sci.*, 24(4):e240406, August 2014. `doi:10.1017/S0960129512000357`.

[EPS73] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, IA*, pages 167–180. IEEE Computer Society, October 1973. URL: `http://dx.doi.org/10.1109/SWAT.1973.11`, `doi:10.1109/SWAT.1973.11`.

[Gad07] Fabio Gadducci. Graph rewriting for the π-calculus. *Math. Struct. Comput. Sci.*, 17(3):407–437, 2007. `doi:10.1017/S096012950700610X`.

[Hin69] J. Roger Hindley. An abstract form of the Church-Rosser Theorem. I. *J. Symb. Log.*, 34(4):545–560, 1969. `doi:10.1017/S0022481200128439`.

[HMP98] Annegret Habel, Jürgen Müller, and Detlef Plump. Double-pushout approach with injective matching. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, November 16-20, 1998, Selected Papers*, volume 1764 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 1998. `doi:10.1007/978-3-540-46464-8\_8`.

[Hof05] Berthold Hoffmann. Graph transformation with variables. In Hans-Jörg Kreowski, Ugo Montanari, Fernando Orejas, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Formal Methods in Software and Systems Modeling, Essays Dedicated to Hartmut Ehrig, on the Occasion of His 60th Birthday*, volume 3393 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2005. `doi:10.1007/978-3-540-31847-7_6`.

[HP96] Annegret Habel and Detlef Plump. Term graph narrowing. *Math. Struct. Comput. Sci.*, 6(6):649–676, 1996.

[HT18] Reiko Heckel and Gabriele Taentzer, editors. *Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig*, volume 10800 of *Lecture Notes in Computer Science*. Springer, 2018. `doi:10.1007/978-3-319-75396-6`.

[JO23] Jean-Pierre Jouannaud and Fernando Orejas. Unification of drags and confluence of drag rewriting. *Journal of Logical and Algebraic Methods in Programming*, 131:100845, February 2023. `doi:10.1016/j.jlamp.2022.100845`.

[LS06] Stephen Lack and Paweł Sobociński. Adhesive categories. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume Lecture Notes in Computer Science, pages 273–288, Berlin, 2006. Springer.

[MM90] José Meseguer and Ugo Montanari. Petri nets are monoids. *Inf. Comput.*, 88(2):105–155, 1990. `doi:10.1016/0890-5401(90)90013-8`.

[MMS97] José Meseguer, Ugo Montanari, and Vladimiro Sassone. Representation theorems for Petri nets. In Christian Freksa, Matthias Jantzen, and Rüdiger Valk, editors, *Foundations of Computer Science: Potential – Theory – Cognition, to Wilfried Brauer on the Occasion of his Sixtieth Birthday*, volume 1337 of *Lecture Notes in Computer Science*, pages 239–249. Springer, 1997. `doi:10.1007/BFb0052092`.

[OE20] Roy Overbeek and Jörg Endrullis. Patch graph rewriting. In *The 13th International Conference on Graph Transformation (ICGT 2020)*, volume 12150 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2020.

[OER21] Roy Overbeek, Jörg Endrullis, and Aloïs Rosset. Graph rewriting and relabeling with PBPO+. In *Proceedings of the 14th International Conference on Graph Transformation (ICGT)*, pages 60–80, Berlin, June 2021. Springer. Held as Part of STAF 2021, Virtual Event. doi:10.1007/978-3-030-78946-6_4.

[PEM86] Francesco Parisi-Presicce, Hartmut Ehrig, and Ugo Montanari. Graph rewriting with unification and composition. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors, *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science, Warrenton, VA*, volume 291 of *Lecture Notes in Computer Science*, pages 496–514. Springer, December 1986. doi:10.1007/3-540-18771-5_72.

[PH94] Detlef Plump and Annegret Habel. Graph unification and matching. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Selected Papers of the 5th International Workshop on Graph Grammars and Their Application to Computer Science, Williamsburg, VA*, volume 1073 of *Lecture Notes in Computer Science*, pages 75–88. Springer, November 1994. doi:10.1007/3-540-61228-9_80.

School of Computer Science, Tel Aviv University, Ramat Aviv, Israel

École Normale Supérieure de Paris-Saclay, France

Universitat Politècnica de Catalunya, Barcelona, Spain