Unification of drags

Jean-Pierre Jouannaud, Fernando Orejas

Unification of drags

Jean-Pierre Jouannaud[1] and Fernando Orejas[2]

1 **Université de Paris-Saclay, Laboratoire de Spécification et Vérification, France**
2 **Universitat Politècnica de Catalunya, Barcelona, Spain**

Abstract

We develop unification for graph rewriting based on the drag model [8].

1 Introduction

Rewriting with graphs has a long history in computer science, graphs being used to represent data structures, but also program structures and even concurrent and distributed computational models. They therefore play a key rôle in program evaluation, transformation, and optimization, and more generally program analysis; see, for example, [2].

Our work is based on a recent, purely combinatorial view of graphs [8]. Drags are labelled graphs equipped with roots and sprouts. Roots are vertices without predecessors that can be seen as input ports, while sprouts are vertices without successors labelled by variables that can be seen as output ports. Drags appear as a generalization of terms which admit many roots, arbitrary sharing, and cycles. Rewrite rules are then pairs of drags that preserve variables and roots, hence avoiding the creation of dangling edges when rewriting. A key aspect of drags is that they can be equipped with a composition operator so that matching a left-hand side of rule L w.r.t. an input drag D amounts to write D as the composition of a context graph C with L, and rewriting D with the rule $L \to R$ amounts to replace L with R in that composition. Since substitutions cannot be separated from context in presence of cycles, composition must play both rôles of plugging a context and a substitution.

To assess our claim that drags are a natural generalization of terms, it is our program to extend the most useful term rewriting techniques to drags. The recursive path ordering is extended in [7]. In this paper, we consider unification. First, we generalize the subsumption order to drags thanks to composition. Our main result is then unifiable drags admit a most general unifier which can be computed in quadratic time. Comparisons with the literature is addressed in Section 2. An interesting relationship between unification of drags and of rational dags is pointed out in conclusion.

2 Related work

The first, dominent model for graph rewriting was introduced in the mid-seventies by Hartmut Ehrig and his collaborators [12]. Referred to as DPO (Double Push-Out), this purely categorical model was then extended in various ways, but also specialized to specific classes of graphs, namely those that do not admit cycles [21]. In particular, termination, unification and confluence techniques have been elaborated for various generalization of trees, such as rational trees, directed acyclic graphs, jungles, term-graphs, lambda-graphs, as well as for graphs in general. See [5] for an old detailed account of these techniques, and [15] for a survey of implementations of various forms of graph rewriting and available analysis tools.

DPO applies to any category of graphs that has pushouts [10]. A rule is a span $L \leftarrow I \rightarrow R$, where I is the *interface* specifying which elements (vertices and edges) from the input graph G matching the left-hand side L by an injective morphism m are preserved by the transformation, the elements in $m(L \setminus I)$ being removed from G while the elements in $R \setminus I$ are added to G. The term DPO refers to the two pushouts generated by the span that define the result of a rewrite step. DPO suffers two drawbacks: applying a rewriting rule fails in case it results in dangling edges, and rules do not have variables.

Categorical approaches are very general, they do apply to many different kinds of graph structures. Besides DPO, the most popular one, they include many variations: matching by a non-injective morphism [10], arbitrary adhesive graph categories [10], single pushout transformation (SPO [11, 22]), or Sesqui-Pushout transformation (SqPO [4]), AGREE [3], and *Hyperedge Replacement Systems* [9]. A detailed comparison of the approach based on drags with all these approaches is not obvious and goes beyond the aims of this paper. Drag rewriting aims at providing a faithful generalization of term rewriting techniques to a certain class of graphs named drags by generalizing to drags all constructions underlying term rewriting, i.e., subterm, substitution, matching, replacement and unification. This is done *constructively* by providing a composition operator for drags which does not pop up in the other, purely descriptive approaches, which aim at describing abstractly subgraph replacement. As a consequence, for a long time, neither graphs nor rules included variables that can be substituted in the transformation process. A recent approach that has some similarities with the drag framework is *port graph rewriting* [13], where graphs include *ports* and *roles*, which, in a way, play a similar rôle as roots and variables in drags. However, the transformation process remains similar to DPO graph rewriting with interfaces [1].

Since most of these general approaches lack variables, most works that study graph unification concentrate in the specific case of *directed acyclic graphs* (dags) that are used to represent terms with shared subterms (see, e.g., [20]). A preliminary attempt to handling variables in graph unification is [19], where variables are used to represent labels equipping the vertices or edges. A quite more general approach is [21], where variables represent hyperedges that may be substituted by *pointed* hypergraphs, but unification is solved there for a very restrictive case only. More recently, Hristakiev and Plump consider graph unification for their graph programming language GP2 [16]. Graphs in GP2 are symbolic graphs whose attributes's values are given by variables satisfying some set of constraints [18]. Variables in this work are not substituted by graphs, but by constrained values.

In contrast, drag variables are real variables as in terms, and drag unification enjoys most general unifiers which are computable in quadratic time. It does not only subsume unification of trees, dags and jungles, but also of rational trees, dags or jungles, as we shall discover in the concluding remarks. These results exploit the fact that the successors of a vertex are ordered and their number is fixed by the symbol labelling that vertex. Relaxing these constraints should be duable, but would blow up the number of most general unifiers.

3 The Drag Model [8]

To ameliorate notational burden, we use vertical bars $|\cdot|$ to denote various quantities, such as length of lists, size of sets or of expressions, and even the arity of function symbols. We use $\varnothing$ for an empty list, set, or multiset, $\cup$ for set and multiset union, as well as for list concatenation, and $\smallsetminus$ for set or multiset difference. We mix these, too, and denote by $K \smallsetminus V$ the sublist of a list K obtained by filtering out those elements belonging to a set V. We will also identify a singleton list, set, or multiset with its contents to avoid unnecessary clutter.

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

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

1. V is a finite set of *vertices*;
2. $R : [p \,..\, p+|R|] \to V$ is a finite list of vertices, called *roots*; $R(p+n)$ refers to the nth root in R; unless otherwise stated, $p = 1$;
3. $S \subseteq V$ is a set of *sprouts*, leaving $V \setminus S$ to be the *internal* vertices;
4. $L : V \to \Sigma \cup \Xi$ is the *labeling* function, mapping internal vertices $V \setminus S$ to labels from the vocabulary Σ and sprouts S to labels from the vocabulary Ξ, writing $v : f$ for $f = L(v)$;
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, $|X(v)| = |L(v)|$).

We use R for both the function itself and its resulting list $[R(1) \,..\, R(n)]$, where $n = |R|$. The labeling function extends to lists, sets, and multisets of vertices as expected.

If $b \in X(a)$, then (a, b) is a directed *edge* with *source* a and *target* b. We also write aXb. The reflexive-transitive closure X^* of the relation X is called *accessibility*. A vertex v is said to be *accessible from* vertex u, and likewise that u *accesses* v, if uX^*v. Vertex v is *accessible* if it is accessible from some root. A drag is *clean* if all its vertices are accessible, *linear* if no two sprouts have the same label. The *subdrag* of a drag D *generated* by a subset W of its accessible vertices is the restriction of D to the vertices accessible from W. Note finally that sprouts may be roots (this is essential for having a nice algebra of drags.)

Terms as ordered trees, sequences of terms (forests), terms with shared subterms (dags) and sequences of dags (jungles [14]) are all particular kinds of clean rooted drags. The drag having no vertex, called the *empty* drag (which is also the empty tree), is clean too.

It will often be convenient to consider roots as specific incoming edges and to identify a sprout with the variable symbol that is its label. Note that sprouts may be roots.

Given a drag $D = \langle V, R, L, X, S \rangle$, we make use of the following notations: $\mathcal{V}er(D)$ for its set of vertices; X_D for its successor function; $\mathcal{R}(D)$ for its roots (list or set, depending on context); $\mathcal{V}ar(D)$ for the set of variables labeling its sprouts; $\mathcal{MV}ar(D)$ for the multiset of those variables; $\mathcal{A}cc(D) = X^*(R)$ for its set of accessible vertices; $|D|$, its *size*, for the number of roots and accessible internal vertices; and $C \oplus D$ for the disjoint union of two drags C, D, the roots of D being renamed from $|\mathcal{R}(C)| + 1$.

3.1 Drag composition

A variable in a drag should be understood as a potential connection to a root of another drag, as specified by a connection device called a *switchboard*. A switchboard ξ is a pair of partial injective functions, one for each drag, whose *domain* $\mathcal{D}om(\xi)$ and *image* $\mathcal{I}m(\xi)$ are a set of sprouts of one drag and a set of positions in the list of roots of the other, respectively.

▸ **Definition 2** (Switchboard). Let $D = \langle V, R, L, X, S \rangle$ and $D' = \langle V', R', L', X', S' \rangle$ be drags. A *switchboard* ξ for D, D' is a pair $\langle \xi_D : S \to [1 \,..\, |R'|]; \xi_{D'} : S' \to [1 \,..\, |R|] \rangle$ of partial injective functions such that

1. $s \in \mathcal{D}om(\xi_D)$ and $L(s) = L(t)$ imply $t \in \mathcal{D}om(\xi_D)$ and $\xi_D(s) = \xi_D(t)$ for all sprouts $s, t \in S$;
2. $s \in \mathcal{D}om(\xi_{D'})$ and $L'(s) = L'(t)$ imply $t \in \mathcal{D}om(\xi_{D'})$ and $\xi_{D'}(s) = \xi_{D'}(t)$ for all $s, t \in S$;
3. ξ is *well-behaved*: it does not induce any cycle among sprouts, using ξ, R, R' relationally:
   $\nexists\ n > 0, s_1, \ldots, s_{n+1} \in S, t_1, \ldots, t_n \in S', s_1 = s_{n+1}.\ \forall i \in [1..n].\ s_i\ \xi_D R' X'^*\ t_i\ \xi_{D'} R X^*\ s_{i+1}$

We also say that $\langle D', \xi \rangle$ is an *extension* of D, and that it is *clean* when D' is.
Assuming $\mathcal{V}er(D) \cap \mathcal{V}er(D') = \mathcal{V}ar(D) \cap \mathcal{V}ar(D') = \varnothing$, we define the disjoint union of extensions as $\langle D, \xi \rangle \oplus \langle D', \xi' \rangle = \langle D \oplus D', \xi \oplus \xi' \rangle$, where $\xi \oplus \xi'(x) = \xi(x)$ if $x \in \mathcal{D}om(\xi)$ and $\xi \oplus \xi'(x) = \xi'(x) + |\mathcal{R}(D)|$ if $x \in \mathcal{D}om(\xi')$.

Sprouts labelled by the same variable should be connected to the same vertex, as required by conditions (1,2). These conditions are of course automatically satisfied by switchboards ξ, called *linear*, defined for sprouts whose variables are all different. It follows that $\xi_D(\mathcal{D}om(\xi_D))$ must be a set, making the set difference $[1..|R'|] \setminus \xi_D(\mathcal{D}om(\xi_D))$ well defined.

We now move to the composition operation on drags induced by a switchboard. The essence of this operation is that the (disjoint) union of the two drags is formed, but with sprouts in the domain of the switchboards merged with the roots to which the switchboard images refer. Merging sprouts with their images requires one to worry about the case where multiple sprouts are merged successively, when the switchboards map sprout to rooted-sprout to rooted-sprout, until, eventually, an internal vertex of one of the two drags must be reached because a switchboard is well-behaved. That vertex is called *target*:

▸ **Definition 3** (Target). Let $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ be drags such that $V \cap V' = \varnothing$, and ξ be a switchboard for D, D'. The *target* $\xi^*(s)$ is a mapping from sprouts in $S \cup S'$ to vertices in $V \cup V'$ defined as follows:

Let $v = R'(n)$ if $s \in S$, and $v = R(n)$ if $s \in S'$, where $n = \xi(s)$.

1. If $v \in (V \cup V') \setminus (S \cup S')$, then $\xi^*(s) = v$.
2. If $v \in (S \cup S') \setminus \mathcal{D}om(\xi)$, then $\xi^*(s) = v$.
3. If $v \in \mathcal{D}om(\xi)$, then $\xi^*(s) = \xi^*(v)$.

The target mapping $\xi^*(\_)$ is extended to all vertices of D and D' by letting $\xi^*(v) = v$ when $v \in (V \setminus S) \cup (V' \setminus S')$.

▸ **Example 4.** Consider the last of the three examples in Figure 1, in which a drag D, whose list of roots is $R = [f\, h\, x]$ (identifying vertices with their label) is composed with a second drag whose list of roots is $R' = [g\, y\, y]$, via the switchboard $\{x \mapsto 3, y \mapsto 2\}$. We calculate the target of the two sprouts: $x\, \xi\, 3\, R'\, y\, \xi\, 2\, R\, h$; hence $\xi^*(x) = \xi^*(y) = h$.

We are now ready for defining the composition of two drags. Its set of vertices will be the union of two components: the internal vertices of both drags, and their sprouts which are not in the domain of the switchboard. The labeling is inherited from that of the components.

▸ **Definition 5** (Composition). Let $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ be drags such that $V \cap V' = \varnothing$, and let ξ be a switchboard for D, D'. Their *composition* is the drag $D \otimes_\xi D' = \langle V'', R'', L'', X'', S''\rangle$, with interface (R'', S'') denoted $(R, S) \otimes_\xi (R', S')$, where

1. $V'' = (V \cup V') \setminus \mathcal{D}om(\xi)$;
2. $S'' = (S \cup S') \setminus \mathcal{D}om(\xi)$;
3. $R'' = \xi^*(R([1\, ..\, |R|] \setminus \xi_{D'}(\mathcal{D}om(\xi_{D'})))) \cup \xi^*(R'([1\, ..\, |R'|] \setminus \xi_D(\mathcal{D}om(\xi_D))))$;
4. $L''(v) = L(v)$ if $v \in V \cap V''$; and $L''(v) = L'(v)$ if $v \in V' \cap V''$;
5. $X''(v) = \xi^*(X(v))$ if $v \in V \setminus S$; and $X''(v) = \xi^*(X'(v))$ if $v \in V' \setminus S$

If ξ_D is surjective and $\xi_{D'}$ total, a kind of switchboard that will play a key rôle for rewriting, then *all* roots and sprouts of D' disappear in the composed drag.

▸ **Example 6.** We show in Figure 1 three examples of compositions. The first is a substitution of terms. The second uses a bi-directional switchboard, which induces a cycle. In that example, the remaining root is the first (red) root of the first drag which has two roots, the first red, the other black. The third example shows how sprouts that are also roots connect to roots in the composition (colors black and blue indicate roots' origin, while red indicates a root that disappears in the composition). Since x points at y and y at the second root of the first drag, a cycle is created on the vertex of the resulting drag which is labelled by h. Further, the third root of the first drag has become the second root of

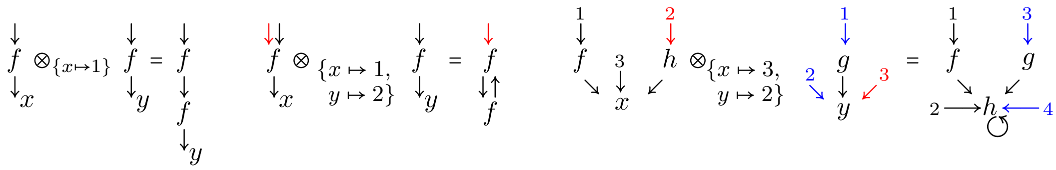

Figure 1 Different forms of composition: substitution, formation of a cycle, and transfer of roots.

the result, while the first (resp., second) root of the second drag has become the third (resp., fourth) root of the result. This agrees of course with the definition, as shown by the following calculations (started in Example 4): $\xi^*([1,2,3]\setminus[2]) = \xi^*([1,3]) = [f,h]$; and $\xi^*([1,2,3]\setminus[2]) = \xi^*([1,2]) = [g,h]$, hence the list of roots of the resulting drag is $[f,h,g,h]$.

The definition of composition does not assume any property of the input drags. Composing a single-rooted clean drag D whose size is at least 1, with a non-clean drag C consisting of a single non-root sprout labelled x, has an observable effect on D: the result of the composition $D \otimes_{\{x\mapsto 1\}} C$ is the drag D', which is D deprived from its root, hence is non-clean since D has internal vertices. Consider now the composition $D \otimes_{\{x\mapsto 1\}} (C \oplus E)$, where E is arbitrary. The result is the drag $D' \oplus E$, hence yields E once cleaned. This is impossible with non-empty clean drags since the size of a drag obtained by composition of two clean drags must be equal to the sum of the sizes of its components.

Composition has important algebraic properties, existence of identities and associativity [6]. We recall the first, and then extend slightly the second.

A clean linear drag all of whose vertices are its sprouts, whose set of edges is empty, and whose list of roots is a list of its sprouts, is called an *identity*. We denote it by 1_X^Y, where X is its set of sprouts and Y is its list of roots. We use $\varnothing$ for the identity empty drag $1_\varnothing^\varnothing$.

We call *identity extension* of a drag D a pair $\langle 1_X^Y, \iota\rangle$ made of:
(i) an identity 1_X^Y such that $\forall x \in \mathcal{V}ar(D)$, D has as many sprouts labelled x as the number of occurrences of $\alpha(x)$ in Y, where $\alpha : \mathcal{V}ar(D) \mapsto X$ is a bijection called *variable renaming*;
(ii) an *identity* switchboard ι satisfying: $\mathcal{D}om(\iota_{1_X^Y}) = \varnothing$, $\mathcal{D}om(\iota_D) = \mathcal{V}ar(D)$; and $\iota_D(x) = \alpha(x)$ (abusing our notations as usual).
Then, the composition $D \otimes_\iota 1_X^Y$ replaces variables of D by those of X, written $D \otimes_\iota 1_X^Y =_\alpha D$.

▸ **Lemma 7** (Neutrality). *Let D be a drag, and $\langle C, \xi\rangle$ a clean extension of D. Then, $C \otimes_\xi D$ and D are identical (up to a variable renaming α) iff $\langle C, \xi\rangle$ is an identity extension $\langle 1_X^Y, \xi\rangle$. Further $C \otimes_\xi D = D$ iff $X = \mathcal{V}ar(D)$ and α is the identity map.*

▸ **Lemma 8** (Associativity). *Let U, V, W be three drags, and ζ, ξ be two switchboards for respectively (V, W) and $(U, V \otimes_\zeta W)$. Then, there exist switchboards θ, γ for respectively (U, V) and $((U \otimes_\theta V), W)$ such that $(U \otimes_\theta V) \otimes_\gamma W = U \otimes_\xi (V \otimes_\zeta W)$.*

Proof. Define $\theta = \langle \xi_{U\to V}, \xi_{V\to U}\rangle$ and $\gamma = \langle \xi_{U\to W} \cup \zeta_{V\to W}, \xi_{W\to U} \cup \zeta_{W\to V}\rangle$. This defines indeed a switchboard since, by definition of ξ and ζ, $\mathcal{D}om(\xi_{U\to W}) \cap \mathcal{D}om(\zeta_{V\to W}) = \mathcal{D}om(\xi_{W\to U}) \cap \mathcal{D}om(\zeta_{W\to V}) = \varnothing$ on the one hand, and $\mathcal{I}m(\xi_{U\to W}) \cap \mathcal{I}m(\zeta_{V\to W}) = \mathcal{I}m(\xi_{W\to U}) \cap \mathcal{I}m(\zeta_{W\to V}) = \varnothing$ on the other hand. The equality then follows straightforwardly. ◂

3.2 Drag rewriting

Rewriting with drags is similar to rewriting with trees: we first select an instance of the left-hand side L of a rule in a drag D by exhibiting an extension $\langle W, \xi\rangle$ such that $D = W \otimes_\xi L$ – this is drag matching, then replace L by the corresponding right-hand side R in the composition. A very important condition for the result to be a drag is, accordingly, that the left- and

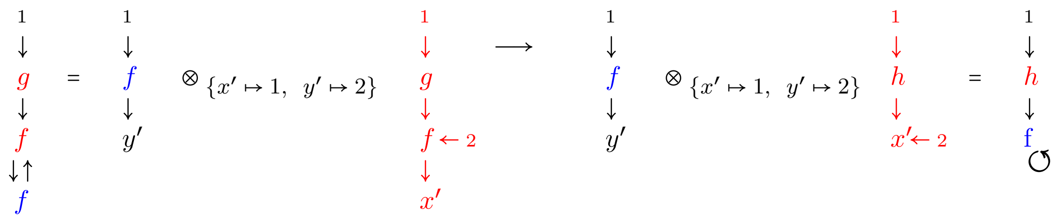

Figure 2 Rewriting and cycles.

right-hand sides of rules have the same number of roots and the same variables in equal number for each. Our definition will therefore always avoid the creation of ill-formed drags.

▸ **Definition 9** (Rules). A *drag rewrite rule* is a pair of clean drags[1], written $L \to R$, such that (i) $|\mathcal{R}(L)| = |\mathcal{R}(R)|$, and (ii) $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$.

Condition (i) ensures that L and R have a perfect fit with any (same) environment – that is, that both can be composed with any extension of L. Condition (ii) is standard for rewrite rules.

Rewriting drags uses a specific kind of switchboard, which allows one to "encompass" a drag U within drag D, so that all roots and sprouts of U disappear from the composition:

▸ **Definition 10** (Rewriting Switchboard). A switchboard ξ for W, U is a *rewriting* switchboard if ξ_W is linear and surjective and ξ_U is total. The pair $\langle W, \xi\rangle$ is a *rewriting extension* of U.

▸ **Definition 11** (Rewriting). Let $\mathcal{R}$ be a graph rewrite system. We say that a nonempty clean drag D *rewrites* to a clean drag D', and write $D \longrightarrow_{\mathcal{R}} D'$, iff $D = C \otimes_\xi L$ and $D' = (C \otimes_\xi R)$ for some drag rewrite rule $L \to R \in \mathcal{R}$ and clean rewriting extension $\langle C, \xi\rangle$ of L.

Because ξ is a rewriting switchboard, ξ_C must be linear, implying that the variables labeling the sprouts of C that are not already sprouts of D must all be different. Then, ξ_C must be surjective, implying that the roots of L (hence those of R) disappear in the composition, a case where the composition is commutative –we shall mostly write the context on the left, though. Further, ξ_L must be total, implying that the sprouts of L (hence those of R) disappear in the composition. Finally, D and C being clean, it is easy to show that D' is clean as well, which is therefore a property rather than a requirement.

▸ **Example 12.** The (red) rewrite rule $g(f(x')) \to h(x')$, whose roots are g and f on the left-hand side and h and x' on the right-hand side, applies with a blue context, colours which are reflected in the input term (showing the rule applies across its cycle) and output term.

We are now finished with the material from [8] needed for the rest of this paper.

4 Unification

The purpose of this section is to *identify* two clean drags U, V by composing them with the same *minimal* rewriting context $\langle C, \xi\rangle$, resulting in the same drag W. An identification

[1] Defining a pattern as a drag all of whose internal vertices are accessible, and assuming that right-hand sides of rules are patterns, it becomes possible to ensure that $\mathcal{MV}ar(L) = \mathcal{MV}ar(R)$. This was exploited in [8] to simplify some technicalities. We prefer the simpler definition here.

corresponds to the fact that we want the same drag to be rewritten by two different rewrite rules whose left-hand sides are U and V. In order for $C \otimes_\xi U$ and $C \otimes_\xi V$ to both make sense, we assume that U, V are *renamed apart*, that is, they share no vertex name, and no root number either. For example, $\mathcal{D}om(\mathcal{R}(U)) = [1..m]$ and $\mathcal{D}om(\mathcal{R}(V)) = [m+1..m+n]$. This will allow us to consider roots as new internal vertices named $r:i$ for $i \in [1..m+n]$, so that vertices are still renamed apart.

In the sequel, U and V will always be clean drags that have been renamed apart.

▸ **Definition 13.** Given drags U, V, we call *partner vertices* two lists L_U, L_V of equal length of internal vertices of U and V, respectively, such that no two vertices $u, u' \in L_U$ (resp., $v, v' \in L_V$) are in relationship with X_U (resp., X_V).

▸ **Definition 14.** Two drags U, V are *identified* with a drag W at *partner vertices* $(\overline{u}, \overline{v})$ by an injective function $o : \mathcal{V}er(U) \cup \mathcal{V}er(V) \to \mathcal{V}er(W)$ called *identification* (extended to lists of vertices in the natural way), which we write $U[\overline{u}] =_o V[\overline{v}]$, iff:

1. $\forall w \in \mathcal{V}er(U), w' \in \mathcal{V}er(V)$ such that $o(w) = o(w')$, $w : f$ iff $w' : f$ iff $o(w) = o(w') : f$;
2. $\forall w \in \mathcal{V}er(U), w' \in \mathcal{V}er(V)$ such that $o(w) = o(w')$, $o(X_U(w)) = o(X_V(w))$;
3. $o(\overline{u}) = o(\overline{v})$.

Note that W may be built from pieces of U and V, since there are no restrictions on the name of its vertices (o may be the identity on appropriate subsets of $\mathcal{V}er(U)$ and $\mathcal{V}er(V)$).

While two terms u, v are unified *at their root*, the solution being a substitution σ such that $u\sigma = v\sigma$, two drags U, V are unified at partner vertices $(\overline{u}, \overline{v})$, the solution being an extension $\langle C, \xi \rangle$ of both U and V that identifies $C \otimes_\xi U$ and $C \otimes_\xi V$ at these partner vertices:

▸ **Definition 15.** A *unification problem* is a pair of connected, clean drags (U, V) that are renamed apart, together with partner vertices $P = \{(\overline{u}, \overline{v})\}$, which we write $U[\overline{u}] = V[\overline{v}]$. A *solution* (or *unifier*) to the unification problem $U[\overline{u}] = V[\overline{v}]$ is a clean rewriting extension $\langle C, \xi \rangle$ such that the *overlap* drags $C \otimes_\xi U$ and $C \otimes_\xi V$ are identified at P. A unification problem $U[\overline{u}] = V[\overline{v}]$ is *solvable* if it has a solution.

▸ **Example 16.** Let $U = f(h(x))$ and $V = f(h(a))$, in which U has two roots, f and h in this order, and V has two roots h and f in this order. These roots are numbered $1, 2, 3, 4$. Let the partner vertices be $\{(h, h)\}$. Then, the corresponding unification problem has for solution the rewriting extension $\langle C, \xi \rangle$ such that $C = a \oplus f(y) \oplus f(z)$, which has three roots, a, f, f in this order, and $\xi = \{x \mapsto 1, y \mapsto 4, z \mapsto 2\}$. The overlap is the drag which has two roots labelled f and f in this order with the drag $h(a)$ being their common successor. Note that flipping the two roots of V would give another solvable unification problem, since the predecessors of a vertex are not ordered (unless they are also successors). Note also that the two root vertices of U and V, which are both labelled by the same function symbol f, remain distinct in the obtained overlap.

Another solution is the extension $\langle a, \{x \mapsto 1\} \rangle$, the overlap being the drag $f(h(a))$. This extension identifies U and V at the partner vertices $\{(f, f)\}$, which is more than needed. ◂

Indeed, we want unification to be minimal, that is, to capture all possible extensions that identify U and V, without useless identifications occuring above or below partner vertices.

In a first subsection, we define the subsumption order on drags (and drag extensions) and show that it is well-founded. This order aims at defining precisely the notion of minimality. In a second subsection, we show that unification of drags is unitary, as for terms and dags.

4.1 Subsumption

▸ **Definition 17.** We say that a drag U is an *instance* of a drag V, or that V *subsumes* U, and write $U \geq V$, if there exists a clean context extension $\langle C, \xi \rangle$ such that $U = C \otimes_\xi V$.

Cleanness is essential here, since otherwise any drag would be an instance of any other drag, which is also the reason why rewriting considers clean extensions only. In the following, we assume for convenience that the sprouts of U, V are labelled by different sets of variables.

▸ **Lemma 18.** *$\geq$ is a quasi-order whose equivalence is variable renaming and strict part is a well-founded order.*

Proof. The relation $\geq$ being reflexive, we show transitivity. Let U, V, W be three drags whose sprouts are labelled by pairwise disjoint sets of variable, such that $U \geq V \geq W$. Then, $U = C \otimes_\xi V$ and $V = D \otimes_\zeta W$, for some clean context extensions $\langle C, \xi \rangle$ of V and (D, ζ) of W. By Lemma 8, $U = E \otimes_\theta W$, for some context extension $\langle E, \theta \rangle$ of W, hence $U \geq W$.

Assume $U \geq V \geq U$. We use the notations above with $U = W$. Let $X = \mathcal{V}ar(U)$ and $Y = \mathcal{V}ar(V)$. Then, $U = E \otimes_\theta U$, hence $\langle E, \theta \rangle$ is an identity extension $\langle 1_X^{X'}, \theta \rangle$ by Lemma 7 and θ the identity switchboard mapping the sprouts labelled by X to the corresponding roots in X'. Hence C, D must be identity drags $1_Y^{Y'}$ and $1_X^{X'}$, and ξ, ζ switchboards mapping respectively X to Y' and Y to X'. Since their composition is the identity, there is a bijection α between $\mathcal{V}ar(U)$ and $\mathcal{V}ar(V)$.

Assume $U > V$. Then, either $|U| > |V|$, or $|U| = |V|$. In the latter case, they cannot have the same number of variables, since otherwise U and V would be identical up to variable names, which contradicts our assumption. Since $U > V$, then $U = C \otimes_\xi V$, where ξ must be an identity drag since U and V have the same size. But ξ cannot be bijective, otherwise $U \simeq V$. Hence ξ maps at least two variables of V to a same (rooted) variable of U. Well-foundedness follows, since the number of variables of a drag of a given size is bounded. ◂

The subsumption quasi-order for drags, despite its name, does not generalize the subsumption quasi-order for terms, which does not take the context into account, but only the substitution. This is however impossible for drags, since the context cannot be separated from the substitution in general (see [6], Decomposition Lemma). Our subsumption quasi-order corresponds therefore to what is called *encompassment* of terms, that is, a subterm of one is an instance of the other. On the other hand, its equivalence generalizes the case of terms, since encompassment and subsumption for terms have the same equivalence.

We now extend the (of course well-founded) subsumption order to context extensions:

▸ **Definition 19.** Let U be a drag, of which $\langle C, \xi \rangle$ and $\langle D, \zeta \rangle$ are two context extensions. We say that (D, ζ) is an *instance* of (C, ξ) (or that (C, ξ) *subsumes* (D, ζ)) w.r.t. U, and write $(D, \zeta) \geq_U (C, \xi)$, if $(D \otimes_\zeta U)$ is an instance of $(C \otimes_\xi U)$.

4.2 Unification algorithm

Since subsumption is well-founded, the set of solutions of a unification problem $U[\overline{u}] = V[\overline{v}]$ has minimal elements when non-empty. What is yet unclear is how to compute them, and whether there are several or one as for terms. This is the problem we address now.

Identifying $C \otimes_\xi U$ and $C \otimes_\xi V$ at a pair of vertices (u, v) requires that u and v have the same label, and that the property can be recursively propagated to their corresponding pairs of successors. To organize the propagation, we shall mark the pair (u, v) with a fresh red natural number before the propagation has taken place (the initial partner vertices will

$$\text{Propagate} \quad U \oplus V \left[\begin{array}{ccc} & u : f \cdot \mathbf{i} & \\ \swarrow & & \searrow \\ s_1 & \dots & s_n \end{array}\right]\left[\begin{array}{ccc} & v : f \cdot \mathbf{i} & \\ \swarrow & & \searrow \\ t_1 & \dots & t_n \end{array}\right] \Rightarrow U \oplus V \left[\begin{array}{ccc} & f \cdot \mathbf{i} & \\ \swarrow & & \searrow \\ s_1 \cdot \mathbf{c+1} & \dots & s_n \cdot \mathbf{c+n} \end{array}\right]\left[\begin{array}{ccc} & f \cdot \mathbf{i} & \\ \swarrow & & \searrow \\ t_1 \cdot \mathbf{c+1} & \dots & t_n \cdot \mathbf{c+n} \end{array}\right]$$

$$\textit{Variable case} \quad U \oplus V[s : x \cdot \mathbf{c}][u : f \cdot \mathbf{c}] \Rightarrow U \oplus V[s : x \cdot \mathbf{c}][u : f \cdot \mathbf{c}]$$

$$\textit{Merge} \quad U \oplus V[s : x][t : x] \Rightarrow U \oplus V[s \cdot \mathbf{c+1}][t \cdot \mathbf{c+1}]$$

$$\textit{Transitivity} \quad U \oplus V[u \cdot \mathbf{i} \cdot \mathbf{j}][v \cdot \mathbf{i}][w \cdot \mathbf{j}] \Rightarrow U \oplus V[v \cdot \mathbf{c+1}][w \cdot \mathbf{c+1}]$$

$$\textit{Symbol conflict} \quad U \oplus V[u : f \cdot \mathbf{i}][v : g \cdot \mathbf{i}] \Rightarrow \bot \ \ \mathbf{if} \ \ f \neq g$$

$$\textit{Internal conflict} \quad U \oplus V[u \cdot i][v \cdot i] \Rightarrow \bot$$

if internal vertices u, v belong both either to U or to V

Occur check

$$U \oplus V[s_1 : x_1 \cdot i_1][w_1 \cdot i_1] \dots [s_n : x_n \cdot i_n][w_n \cdot i_n] \Rightarrow \bot$$

$$\mathbf{if} \begin{cases} \forall i \in [1..n] \ s_i \text{ is a sprout and } w_i \text{ is an internal vertex} \\ \forall i \in [1..n] \ s_{i+1} \text{ (convention: } s_{n+1} = s_1\text{) is accessible from } w_i \\ \exists i \in [1..n] \ w_i \text{ is not rooted} \end{cases}$$

Figure 3 Drag unification rules

hold marks $\mathbf{1}, \dots |\overline{\mathbf{u}}|$), and turn this mark into blue once the propagation has taken place. In case one of u, v is a sprout, no propagation occurs, it is enough to turn the red mark into blue. To ensure freshness, we shall memorize the number of pairs of vertices that have been marked so far. Our syntax for marking a vertex is as in $u : f \cdot \mathbf{i_1} \cdots \mathbf{i_n}$ if u has label f and (blue or red) marks $\mathbf{i_1}, \dots, \mathbf{i_n}$. Vertex u, label f or marks may be omitted when convenient.

Propagation stops when there are no more pairs of internal vertices holding a red mark, unless two sprouts hold the same variable, in which case they must now be marked in red if not marked already, giving possibly rise to a new round of propagations.

We call c the number of already used marks, initialized to 0, and incremented by one at each use of a mark, including the initial marking of the partner vertices. The rules are reminiscent from the unification rules for terms, see for example [6], although we don't use the same rule names except for *Merge* and *Occur check*. For example, we use *Propagate* rather than *Decompose* to stress the fact that drags cannot be treated as terms.

We single out a vertex w in the drag $U = V$ by writing $U = V[w]$, a notation that extends to several, possibly identical, vertices as expected. We assume that vertices singled out in left-hand sides are pairwise different, and that a pair of vertices sharing a mark is not marked again. It follows that the configurations $U = V[u \cdot i \cdot j][v \cdot i \cdot j]$ and $U = V[u \cdot i \cdot i]$ cannot occur.

The procedure described at Figure 3 computes an equivalence relation between the vertices of two drags to be unified from which their most general unifier is extracted in Section 4.4. It consists in a set of transformation rules operating on the unification problem $U = V$ (actually, on the drag $U \oplus V$) by marking pairs of vertices with elements of an initial

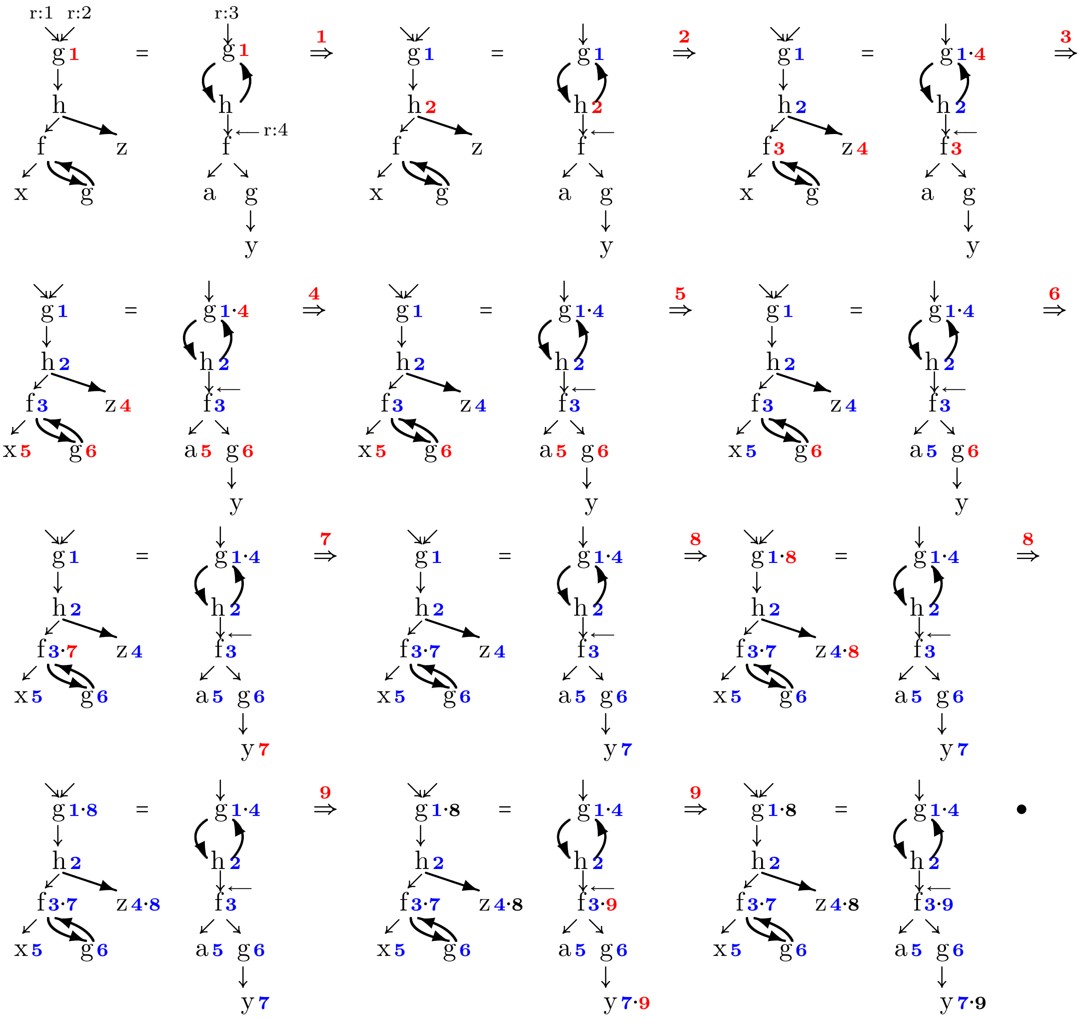

Figure 4 Successful unification of two patterns.

segment of the natural numbers, figuring out new edges of a specific kind between vertices of $U \oplus V$, in the style of Patterson and Wegman unification algorithm [20]. A related idea appears also in [17].

▸ **Example 20.** In our example of Figure 4, unification of the initial two drags proceeds in eleven steps and succeeds. *Propagation* steps are labelled by the red mark processed while *Transitivity* steps are labelled by the generated mark. This explains why some steps have the same label. Unifying these drags at roots r:1 and r:3 would increment all marks by 1.

A failure case is carried out at Figure 7, displayed at the very end of the paper, which proceeds in two rounds. This time, there is a variable shared between both drags.

Note that roots are not part of equivalence classes, unless they belong to the list of partner vertices: since they do not have predecessors, they are never marked unless as partner vertices, in which case they are then treated as internal vertices.

An important, immediate property of the unification rules is termination:

▸ **Lemma 21.** *Unification rules terminate in quadratic time w.r.t. the size of the input.*

Proof. Since a pair of identical vertices is never maked, and a pair of different vertices is never marked twice, the number of marks of a unification problem $U[u] = V[v]$ is at most equal to $(|U| + |V|) \times (|U| + |V| - 1)$. ◂

4.3 Soundness of the unification rules

Soundness is the property that the solutions of a unification problem $U[\overline{u}] = V[\overline{v}]$ are preserved by application of the unification rules, until some normal form is obtained.

Propagations reveal dependencies among the vertices of two unifiable drags which lead to the central notion of *generated equivalence* between their vertices.

▸ **Definition 22.** An equivalence $\equiv$ on the set of vertices of a drag $U \oplus V$ is *generated* by partner vertices $(\overline{u}, \overline{v})$ iff (i) vertices in $\overline{u}, \overline{v}$ are pairwise equivalent; (ii) any two equivalent internal vertices u, v have identical labels; (iii) the successors of equivalent internal vertices are pairwise equivalent; and (iv) sprouts with identical label are equivalent.

The notion of generated equivalence corresponds to that of a congruence on terms.

Notation: we denote by $U[\overline{u}] =_k V[\overline{v}]$, or simply $U =_k V$, the unification problem obtained at step k of rewriting the initial unification problem $U[\overline{u}] = V[\overline{v}]$, also denoted by $U =_0 V$.

▸ **Definition 23.** Given a marked unification problem $U =_k V$, we denote by $\equiv_k$ the least equivalence on the vertices of the drag $U \oplus V$ generated by all pairs of vertices that share a common mark. When unification succeeds, we define *unification equivalence* $\equiv_{unif}$ as $\bigcup_k \equiv_k$.

The rules show that vertices of $U[\overline{u}] =_k V[\overline{v}]$ are marked by natural numbers in $[1..k]$. Since the unification rules never remove markings, $\equiv_k$ is monotonically increasing with k:

▸ **Lemma 24.** $\equiv_k \subseteq \equiv_l$ *for all* $l \geq k$ *such that* $\equiv_l$ *is defined.*

It follows that $\equiv_{unif}$ is the equivalence associated with $U =_n V$, the obtained normal form of $U =_0 V$ at step n. We believe that this normal form is unique, a property not needed here.

The property of sharing a mark is clearly reflexive and symmetric. It is also transitive, as shown in Appendix, hence coincides with $\equiv_{unif}$:

▸ **Lemma 25.** *Assume* $u \equiv_{unif} v$*, where* $u \neq v$*. Then, there exists* i *such that* $U =_k V[u{\cdot}i][v{\cdot}i]$ *for some* k *and* i*.*

Proof. By definition of $\equiv_{unif}$, $u \equiv_n v$ for some n. We prove the property by induction on the length of the proof that $u \equiv_n v$. If $U =_n V[u \cdot i][v \cdot i]$, we are done. Otherwise, $u \equiv_n w$ and $U =_n V[w{\cdot}i][v{\cdot}i]$ for some vertex $w \neq u$. By induction hypothesis, $U =_l V[u{\cdot}j][w{\cdot}j]$ for some l and j. By Lemma 24, $U =_p V[u \cdot j][w \cdot j \cdot i][v \cdot i]$ for some p, hence $U =_q V[u \cdot k][v \cdot k]$ for some k, q by *Transitivity.* ◂

▸ **Lemma 26.** $\equiv_{unif}$ *is an equivalence on the vertices of* $U[\overline{u}] \oplus V[\overline{v}]$ *generated by* $(\overline{u}, \overline{v})$*.*

Proof. Initialization ensures that $\overline{u} \equiv \overline{v}$. Since unification has terminated with success, *Propagation* ensures the first three properties of an equivalence, and *Merge* the fourth. ◂

▸ **Lemma 27.** *Let* u, v *be two vertices of* $U \oplus V$ *such that* $u \equiv_{unif} v$*. Then,* u, v *are identified by any solution of* $U = V$*.*

Proof Since $\equiv_{unif} = \bigcup_k \equiv_k$, we prove that the property holds for all k, by induction on k first, then on the definition of $\equiv_k$.

- $u \equiv_0 v$. Then, u, v are the two initial internal partner vertices, which must be identified by definition of a solution of $U[u] = V[v]$.

- $u =_{k+1} v$. If $u =_k v$, we conclude by induction on k. Otherwise, there are two cases:
  - if u, v are successors or predecessors of some pair of vertices u', v' such that $u' =_k v'$. By induction hypothesis, u', v' must be identified. Since $\equiv_{unif}$ is an equivalence by Lemma 26, u, v must be identified as well.
  - Otherwise, $u =_k w =_k v$. We conclude this case by induction hypothesis and transitivity of identification. ◂

We can now show soundness of the rules:

▸ **Lemma 28.** *Let $U = V[s_1 : x_1 \cdot i_1][w_1 \cdot i_1] \dots [s_n : x_n \cdot i_n][w_n \cdot i_n]$ such that (i) $\forall i \in [1..n]$, s_{i+1} is accessible from the internal vertex w_i (with $s_{n+1} = s_1$) and (ii) some w_i has no root. Then, $U = V$ has no solution.*

Proof. Assume that the equation $U = V$ has a solution $\langle C, \xi \rangle$ which therefore identifies vertices having the same mark. Then, $s_1 \otimes_\xi C = w_1 \otimes_\xi C$. Since w_1 is an internal vertex by assumption, the respective numbers of internal vertices accessible from $\xi(s_i)$ and $\xi(s_{i+1})$ in the drag $(U = V) \otimes_\xi C$ must be different, unless there is a cycle going through $\xi(s_i)$. This is only possible if $\xi(s_i)$ is a root of w_i in $U \oplus V$, which contradicts assumption (ii). Therefore, the equation $U = V$ cannot have a solution. ◂

▸ **Lemma 29.** *An internal conflict has no solution.*

Proof. By lemma 27, u and v should be identified, but no clean extension can identify two different internal vertices of a drag. ◂

▸ **Lemma 30.** *Unification rules are sound.*

Proof. Soundness of *Internal conflict* and *Occur check* is shown at Lemmas 29, 28. Soundness of all other rules is a consequence of equality of drags. ◂

4.4 Completeness of the unification rules

Completeness is the property that the unification rules must return an equivalence from which a most general unifier can be constructed.

▸ **Definition 31.** Let $\equiv$ be an equivalence on the vertices of a drag D. We define the *occur-check* relationship on the variables labelling its sprouts as $x >_{oc} y$ if $s : x \equiv u X_D^+ t : y$ for some internal vertex u and sprouts s, t. We say that x is an *occur-check variable* if $x >_{oc}^+ x$, in which case the class M such that $s : x \in M$ is said to be in *occur-check*.

▸ **Definition 32** (Solved form)**.** Given two drags U, V an equivalence $\equiv$ on $U \oplus V$ generated by their partner vertices $(\overline{u}, \overline{v})$ is in *solved form* iff the following properties hold:

- no two internal vertices of U (resp., V) are equivalent;
- Every occur-check variable is equivalent to some rooted internal vertex.

The equivalence classes of a generated equivalence in solved form can therefore contain any number of variables, but at most one internal vertex from each drag U, V.

▸ **Example 33** (Continued)**.** The marked unification problem obtained at Figure 4 gives the following unification equivalence: $1 \equiv 4 \equiv 8$; $3 \equiv 7 \equiv 9$; 5; 2; 6. Note that each class has at most two internal vertices. Variable x belongs to the third class, it is not an ccur-check variable. Variables z and y belonging to the first two classes are both occur-check variables. We oberve that the existence of an appropriate rooted vertex in their class is satisfied for both z (vertex labelled by g with mark $1 \cdot 8$) and y (vertex labelled by f with mark $3 \cdot 9$).

In the sequel, we say that an equivalence class is *non-trivial* when it is not reduced to a singleton.

▸ **Lemma 34.** *The equivalence $\equiv_{unif}$ defined on the vertices of $U[\overline{u}]{\oplus}V[\overline{v}]$ is in solved form.*

Proof We show that a rule applies to unification problems which are not in solved form.

1. By lemma 26, $\equiv_{unif}$ is an equivalence generated by $(\overline{u},\overline{v})$.
2. Let $U \oplus V[u][v]$ such that $u \equiv_{unif} v$, $u, v \in U$ and $u \neq v$. By Lemma 25, $U = V[u \cdot i][v \cdot i]$ for some i, hence *Internal Conflict* applies.
3. Assume $\equiv_{unif}$ does not have enough roots. Then, by Lemma 25, *Occur check* applies. ◂

To show that solved forms have a most general unifier, we define their unifying extension using the operation $\oplus$ between rewrite extensions with its obvious meaning (Definition 2). More precisely, given a unification problem, $U[\overline{u}] = V[\overline{v}]$, and an equivalence on $U \oplus V$ in solved form, $\equiv$, the recursive definition $mgu(U[\overline{u}] \oplus V[\overline{u}], \equiv)$ constructs an extension, (C,ξ), together with an identification function $o : \mathcal{V}er(U) \cup \mathcal{V}er(V) \to \mathcal{V}er(W)$, such that (C,ξ) is a solution to $U[\overline{u}] = V[\overline{v}]$ and $U \downarrow_{\equiv} V = o((U \oplus V) \otimes_{\xi} C)$ is the resulting unified drag.

▸ **Definition 35** (mgu)**.** The *most general unifying extension* $mgu(U[\overline{u}]{\oplus}V[\overline{u}], \equiv) = (C,\xi)$ of the problem $U[\overline{u}] = V[\overline{v}]$ w.r.t. the solved form $\equiv$, and the associated identification function $o : \mathcal{V}er(U) \cup \mathcal{V}er(V) \to \mathcal{V}er(W)$ defining the *unified drag* $U \downarrow_{\equiv} V = o((U \oplus V) \otimes_{\xi} C)$, are defined by induction on the number of non-trivial equivalence classes of $\equiv$ as follows:

- **base case:** if $\equiv$ has no non-trivial equivalence class containing a variable, then the empty extension, $mgu(U \oplus V, \equiv) = \langle\varnothing,\varnothing\rangle$, is already a solution to $U[\overline{u}] = V[\overline{v}]$, and its associated identification o is defined $o(v) = o(w)$ for all u, v such that $u \equiv v$.
- **induction step:** let M be a non-trivial equivalence class of vertices containing at least one variable. Then, $mgu(U{\oplus}V, \equiv) = (C,\xi){\oplus}mgu((U{\oplus}V){\otimes_\xi}C, (\equiv \setminus M) \cup \{o(M)\})$, where:
  1. if M includes two or more variables, but no internal vertices, $M = \{s_1 : x_1, \dots s_n : x_n\}$, with $n > 1$, then C consists of just one sprout s, labelled with some fresh x, equipped with n roots $[1 \dots n]$, $\forall i {\in} [1..n]\ \xi(s_i) {=} i$, and $o(s_i) {=} s{:}x$.
  2. if $M = \{s_1 : x_1, \dots s_n : x_n\} \cup N$, where N is a non-empty set of at most two internal vertices $\{v, w\}$ and $n \geq 1$, and M is not in occur-check, then C is the subdrag of $U{\oplus}V$ generated by v equipped with n roots $[1 \dots n]$, and $o(s_i) = o(v) = o(w) = v$.
  3. if $M = \{s_1 : x_1, \dots s_n : x_n\} \cup N$, where N is a non-empty set of at most two internal vertices $\{v, w\}$ and $n \geq 1$, M is in occur-check, and v has root $r : k$, then C is the identity drag $1_z^{z^n}$ for some fresh variable z, hence is the variable z equipped with n roots $[1 \dots n]$, $\xi(x_i) = i$, $\xi(z) = k$, and $o(x_i) = o(v) = o(w) = v$.

▸ **Example 36** (Continued)**.** Figure 5 shows the context drag, switchboard, and overlapping drag obtained by composition of the inputs drags of Figure 4. The equivalence on vertices considered here is defined by having equal markings, which is in solved form as already observed. Note that unifying at vertices $(r{:}1, r{:}3)$ instead would give the same result. We now carry out the calculation of the unifying extension as specified by Definition 15:

There are three non-trivial equivalence classes containing one variable each. We start with the non-trivial equivalence class containing the variable x which is not in occur-check, which corresponds to Case 2 of the recursive call. We get C is reduced here to the term a whose root name will be 1, and ξ will map x to 1 (abusing our notations as usual), hence $(C,\xi) = (a, x \mapsto 1)$. Since the previous equivalence class is now trivialized in $(\equiv \setminus M) \cup \{o(M)$, the drag $(U \oplus V) \otimes_{\xi}$ has two equivalences classes left to be processed, which we consider in turn. In the first, z is an occur-check variable, and the vertex $g\ 1 \cdot 8$ has two available roots. We therefore apply Case 3 of Definition 15. Which ever available root is chosen does

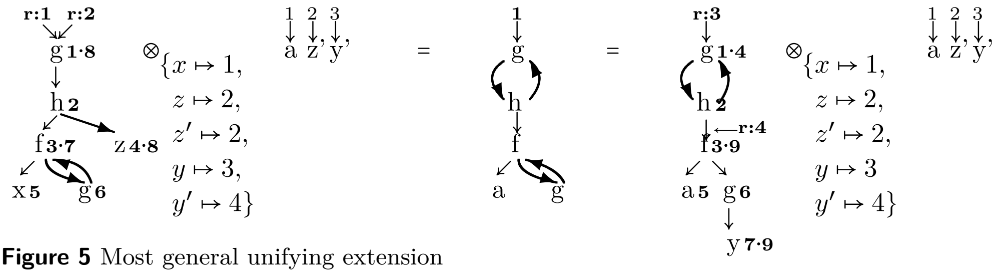

Figure 5 Most general unifying extension

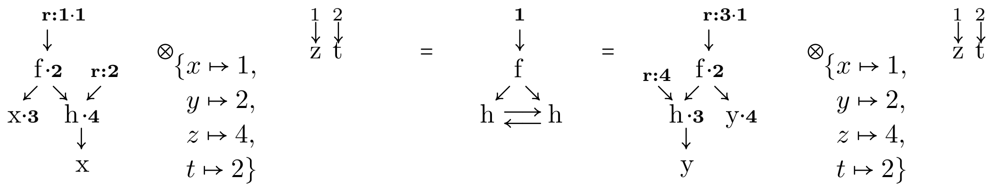

Figure 6 Building a most general unifier from a solved congruence

not impact the rest of the computation, we choose $r:2$ (had we unified at partner vertices $r:1$ and $r:3$ would yield the same result but no choice !). In this case, C is the identity drag $1^{z'}_{z'}$, $\xi(z) = 1$ (the sole root of the identity drag), and $\xi(z') = 1$ (the available root from U that has been chosen). Finally, the same is done for the last class that contains the occur-check variable y. We get here $C = 1^{y'}_{y'}$, $\xi(y) = 1$ and $\xi(y') = 4$. All classes being now trivialized, we are done. We can therefore compose the recursive call backwards to form the unifying context extension, starting with the extension $(1^{y'}_{y'}, \{y \mapsto 1, y' \mapsto 4\})$ obtained at the last call. Composing this extension with the previously obtained one gives $(1^{z'}_{z'}, \{z \mapsto 1, z' \mapsto 2\}) \oplus (1^{y'}_{y'}, \{y \mapsto 1, y' \mapsto 4\})$, and composing with the first obtained extension yields the result $(a, x \mapsto 1) \oplus (1^{z'}_{z'}, \{z \mapsto 1, z' \mapsto 2\}) \oplus (1^{y'}_{y'}, \{y \mapsto 1, y' \mapsto 4\})$. We are left calculating the result according to the definition of $\oplus$. In this process, the root numbers of the context get changed, and accordingly the root numbers of the switchboard ξ for the variables y and z that belong to the input drags, hence take their values in the roots of the context extension, as stipulated by the definition of $\oplus$ for extensions given at Definition 2. The root number of the sprout z' becomes therefore $1 + 1 = 2$, while that of y' becomes $1 + (1 + 1 = 2) = 3$.

The rôle of the extension $\langle C, \xi \rangle$ is to unify $U[\overline{u}]$ and $V[\overline{v}]$. Before any identification has taken place, the drag is $U[\overline{u}] \oplus V[\overline{v}]$. At each recursive call, some new vertices of $U[\overline{u}]$ and $V[\overline{v}]$, which are successors of $\overline{u}, \overline{v}$, become identified in the drag $(U \oplus V) \otimes_\xi C$ while the equivalence class M of $\equiv$ collapses to the same vertex $o(M)$ of the obtained drag. This is why this constructive definition is by induction over the number of non-trivial equivalences classes of $\equiv$. In the base case, there is no need for an extension, we only have to define o. In Case 2, any drag isomorphic to the drag generated from v would do for C. Case 1 is similar, but C has no internal vertex. The occur-check Case 3 can be solved thanks to the roots $r:k$. Note that vertices which are not accessible from $\overline{u}, \overline{v}$ are not identified.

▸ **Example 37.** Figure 6 shows an indirect way of building cycles in a unified drag, whose solved congruence is obtained here by unifying the input drags at the pair of roots $(r:1, r:3)$. Note that the lefthand side h vertex of the unified drag *is* the h vertex of the righthand side input drag while its righthand side h vertex *is* the h vertex of the lefthand side input drag.

▸ **Lemma 38.** *Let* $\equiv$ *be an equivalence in solved form for the unification problem* $U[\overline{u}] = V[\overline{v}]$*. Then, the most general unifying extension* $mgu(U[\overline{u}] \oplus V[\overline{v}], \equiv)$ *is well-defined and is a unifier of* $U[\overline{u}] = V[\overline{v}]$*,* $U \downarrow_{\equiv} V$ *being the resulting unified drag.*

Proof. The proof of these claims is by induction on the number of non-trivial equivalence classes. For the base case, the respective subdrags of U and V generated by $\overline{u}$ and $\overline{v}$ must be identical, up to the name of their vertices, which justifies our three claims in that case.

For the induction step, we first prove that the recursive call $mgu(U \oplus V) \otimes_\xi C, (\equiv \setminus M) \cup \{o(M)\})$ makes sense, that is, that $(\equiv \setminus M) \cup \{o(M)\}$ is a solved form for the drag $(U[u] \oplus V[v]) \otimes_\xi C$ which has strictly less non-trivial classes than $\equiv$. Since it is the drag $U[u] \oplus V[v]$ in which all vertices belonging to the equivalence class M have been identified to $o(M)$, $(\equiv \setminus M) \cup \{o(M)\}$ is the restriction of $\equiv$ to that drag. Since the class M has become trivial, it therefore has stricly less non-trivial equivalence classes. The fact that $\equiv$ is a solved form is easily lifted to $(\equiv \setminus M) \cup \{o(M)\}$ in all three cases since the new equivalence is obtained by identifying all vertices of the sole equivalence class M. We are left showing that the definition delivers a unifier in all three cases, knowing by induction hypothesis that this is true of the recursive call, whose result is the extension $\langle C', \xi' \rangle$ together with the identification o'. By induction hypothesis, $mgu((U \oplus V) \otimes_\xi C, (\equiv \setminus M) \cup \{o(M)\})$ identifies the drag $U \oplus V) \otimes_\xi C$, that is the drag $U \oplus V$ in which the class M has been collapsed to a single vertex by $\langle C, \xi \rangle$. We are therefore left to showing that $\langle C, \xi \oplus mgu((U \oplus V) \otimes_\xi C, (\equiv \setminus M) \cup \{o(M)\})$ identifies $U \oplus V$. This includes showing that $\langle C, \xi \rangle$ is well-defined.

In all cases of the induction step of Definition 35, $\langle C, \xi \rangle$ identifies all vertices of M. In case there is a choice between two internal vertices $\{v, w\}$, whatever choice fulfilling the requirements yields the same drag $(U \oplus V) \otimes_\xi C$, up to the renaming of one vertex, namely $o(M)$. We are left proving that the resulting mapping o is an identification, which is a consequence of the fact that $\equiv$ is a generated equivalence. ◂

▸ **Theorem 39.** *Let* $\equiv$ *be an equivalence in solved form for the unification problem* $U[\overline{u}] = V[\overline{v}]$*. Then,* $mgu(U \oplus V, \equiv)$ *is a most general unifier.*

Proof. By Lemma 38, $\langle C, \xi \rangle = mgu(U[\overline{u}] \oplus V[\overline{v}], \equiv)$ is a unifier of the equation $U[\overline{u}] = V[\overline{v}]$. Hence $U \otimes_\xi C = V \otimes_\xi C$. Let now $\langle D, \zeta \rangle$ be an arbitrary unifying extension of $U[\overline{u}] = V[\overline{v}]$, hence $U \otimes_\zeta D = V \otimes_\zeta D$. We show that $\langle D, \zeta \rangle$ is an instance of $\langle C, \xi \rangle$, more precisely that

$$U \otimes_\zeta D = (U \otimes_\xi C) \otimes_\zeta D,$$

by induction on the construction of $\langle C, \xi \rangle$. The property is satisfied in the base case, since mgu is then an identity. We now show that it is satisfied in all three cases of the construction:

1. Case 1: Since D must be a solution of the solved form, $x_i \otimes_\zeta D = U[u] \otimes_\zeta D$. But $U[u] = x_i \otimes_\xi C$, hence $x_i \otimes_\zeta D = (x_i \otimes_\xi C) \otimes_\zeta D$.
2. Case 2: This case is similar to the above one.
3. Case 3. In this case, both extensions must do exactly the same, hence are identical.

We then conclude by induction in all three cases. ◂

4.5 Most general unifiers

We can therefore end up this section with our main result:

▸ **Theorem 40.** *Unification is unitary, and has quadratic worst case complexity.*

Proof. Follows from Lemmas 30, 21, and 34, and Theorem 39. ◂

5 Conclusion

Drags appear to be an extremely handy generalization of terms, dags and jungles: the intuitions behind them all are very similar, as well as the most important algorithms for implementing rewriting and testing its termination and confluence, despite the possibility of having arbitrary cycles in drags. This is made possible by a powerful composition operator.

Drags do not exactly generalize terms, though, as is pointed out in [8]. This is because our definition of composition *forces* sharing, as does term rewriting in practice. Capturing the term case requires using drag isomorphism instead of drag equality in presence of non-linear variables in rules. This is of course possible, and is currently being investigated.

So drags generalize dags (and jungles), by allowing for cycles. Unwinding these cycles yields infinite dags with finitely many subdags, that is, rational dags. The difference is that the ability to share a subdag requires that a context can always add an incoming edge to that subdag, which is not possible with drags, for which this subdag must be rooted beforehand. So, dags are drags equipped with roots at all vertices that need to be shared, making it then possible to build cycles by composition with a rewrite extension. Therefore drag unification must be at least as complex as rational dags unification, and therefore as rational tree unification, shown almost linear by Huet, whose algorithm actually solves without saying the rational dags unification problem [17]. It may well be that the exact complexity of dag and drag unification are the same, but we have not investigated it yet.

Warm thanks to Anne Yenan and José Motos who provided a *deluxe roof* to the first author during his one month stay in Barcelona at the invitation of the second author.

References

1 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.

2 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.

3 Andrea Corradini, Dominique Duval, Rachid Echahed, Frédéric Prost, and Leila Ribeiro. AGREE - algebraic graph rewriting with controlled embedding. In *Graph Transformation - 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*, pages 35–51, 2015.

4 Andrea Corradini, Tobias Heindel, Frank Hermann, and Barbara König. Sesqui-pushout rewriting. In *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17-23, 2006, Proceedings*, pages 30–45, 2006.

5 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. Elsevier, 1990.

6 Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier, 1990.

7 Nachum Dershowitz and Jean-Pierre Jouannaud. Graph path orderings. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on*

Logic for Programming, Artificial Intelligence and Reasoning, volume 57 of *EPiC Series in Computing*, pages 307–325. EasyChair, 2018.

8 Nachum Dershowitz and Jean-Pierre Jouannaud. Drags: A compositional algebraic framework for graph rewriting. *Theor. Comput. Sci.*, 777:204–231, 2019.

9 Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement, graph grammars. In Rozenberg [23], pages 95–162.

10 Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of algebraic graph transformation.* Springer-Verlag, 2006.

11 Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation – part II: single pushout approach and comparison with double pushout approach. In Rozenberg [23], pages 247–312.

12 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.

13 Maribel Fernández, Hélène Kirchner, and Bruno Pinaud. Strategic port graph rewriting: an interactive modelling framework. *Mathematical Structures in Computer Science*, 29(5):615–662, 2019.

14 Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. *Fundam. Inform.*, 15(1):37–60, 1991.

15 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.

16 Ivaylo Hristakiev and Detlef Plump. A unification algorithm for GP 2 (long version). *CoRR*, abs/1705.02171, 2017.

17 Gérard Huet. *Unification dans les langages d'ordre 1, . . . , ω.* PhD thesis, Université Paris 7, Paris, France, 1976.

18 Fernando Orejas and Leen Lambers. Symbolic attributed graphs for attributed graph transformation. *ECEASST*, 30, 2010.

19 Francesco Parisi-Presicce, Hartmut Ehrig, and Ugo Montanari. Graph rewriting with unification and composition. In *Graph-Grammars and Their Application to Computer Science, 3rd International Workshop, Warrenton, Virginia, USA, December 2-6, 1986*, pages 496–514, 1986.

20 Mike Paterson and Mark N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.

21 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.

22 Jean-Claude Raoult. On graph rewritings. *Theor. Comput. Sci.*, 32:1–24, 1984.

23 Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

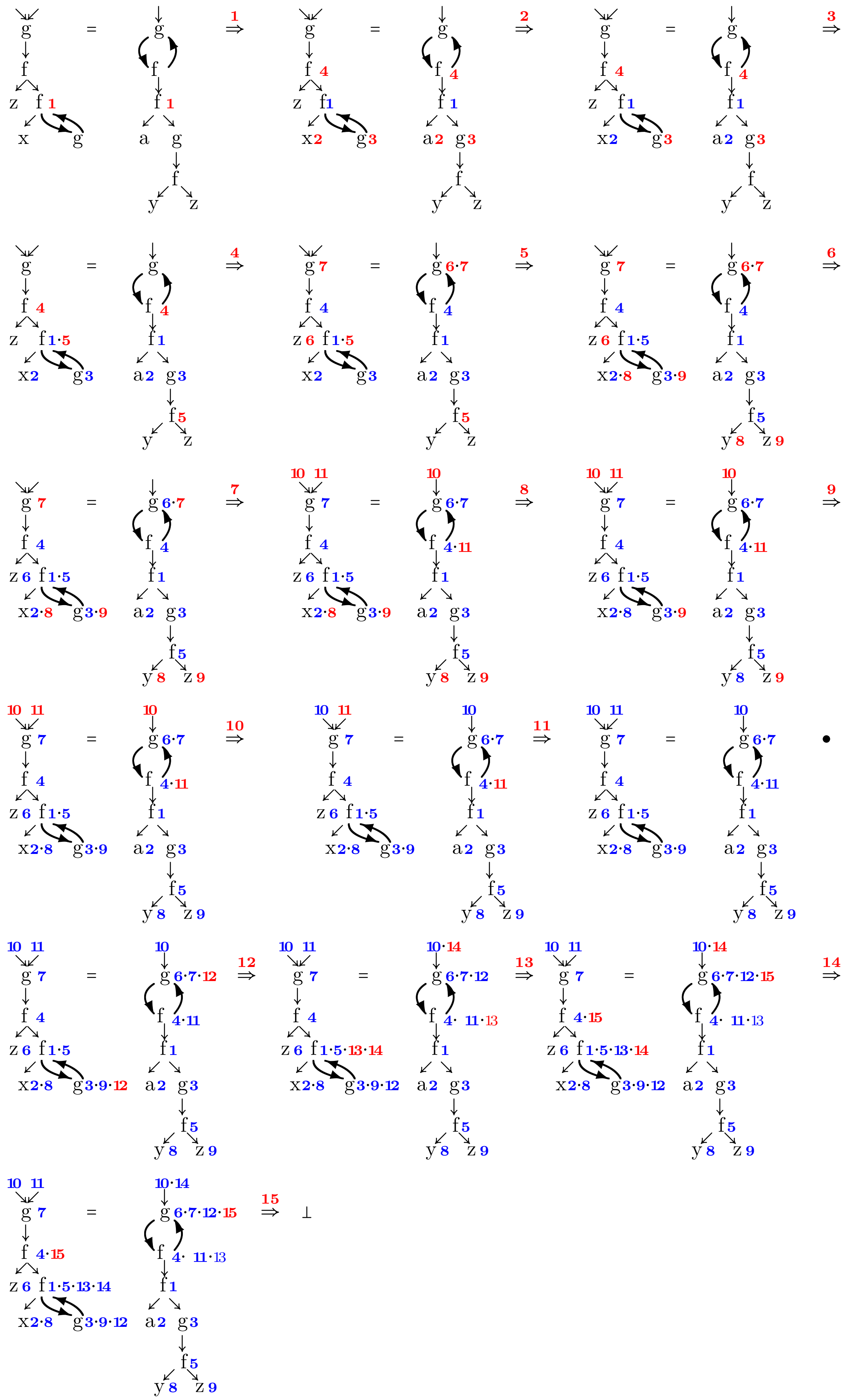

Figure 7 Unification of two patterns: failure case