Unification of Drags and Confluence of Drag Rewriting

Jean-Pierre Jouannaud, Fernando Orejas

myjournal manuscript No.
(will be inserted by the editor)

Unification of Drags and Confluence of Drag Rewriting

Jean-Pierre Jouannaud · **Fernando Orejas**

Submitted: by Dec 14. Received: date / Accepted: date

Abstract Drags are a recent, natural generalization of terms which admit arbitrary cycles. A key aspect of drags is that they can be equipped with a composition operator so that rewriting amounts to replace a drag by another in a composition. In this paper, we develop a unification algorithm for drags that allows to check the local confluence property of a set of drag rewrite rules.

Keywords Drags · graphs · unification · confluence

PACS PACS code1 · PACS code2 · more

Mathematics Subject Classification (2010) MSC code1 · MSC code2 · more

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, [3].

Our work is based on a recent, purely combinatorial view of labeled graphs [9]. Drags are labelled graphs equipped with roots and *sprouts*, which are vertices without successors labelled by variables. Drags appear as a generalization of terms, they admit roots at arbitrary vertices, sharing, and cycles. Rewrite rules are then pairs of drags that preserve variables and number of 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. Composition plays the rôle of both context grafting and substitution in the case of terms.

J.-P. Jouannaud
École Normale Supérieure Paris-Saclay, Laboratoire de Spécification et Vérification, Saclay, France E-mail: jeanpierre.jouannaud@gmail.com

F.Orejas
Universitat Politècnica de Catalunya, Department of Software, Barcelona, Spain

To assess our claim that drags are a natural generalization of terms, we extend the most useful term rewriting techniques to drags: the recursive path ordering [8], unification (Section 3) and local confluence (Section 4). Our first main result here is that unification is unitary and can be performed in quadratic time and space, a complexity bound which is not shown to be sharp and is possibly not. In the case of terms, unification is based on overlapping two terms at a non-variable subterm, from which a recursive propagation process takes place that identifies the labels of both term fragments as long as no label is a variable. The unification process for drags is similar, starting at a set of partner vertices at which the overlap takes place. Propagation operates on pairs of vertices which have not been propagated yet provided no vertex in a pair is a sprout. Our second main result is that local confluence of a set of drag rewrite rules can be checked by the usual joinability test of their critical pairs. This is so because local confluence follows easily in the non-overlap case since the rewritten drag is then the composition of two drags that are both rewritten independently of each other. The so-called disjoint and ancestor cases that pop up in the case of terms are therefore both captured here by the same case, hence showing the advantage of packaging context grafting and substitution within a single composition mechanism. As a result, confluence is decidable for terminating finite sets of drag rewrite rules, as is the case for term rewrite rules. Comparisons with the literature is addressed in Section 5. An interesting relationship between unification of drags and of rational dags is pointed out in conclusion.

2 The Drag Model [9]

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 $\setminus$ for set or multiset difference. We mix these, too, and denote by $K \setminus V$ 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, $|X(v)| = |L(v)|$).

The pair (R, S) will be called the *interface* of the drag D.

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, denoted by $D_{|W}$, 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 [18]) 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: $\mathscr{V}er(D)$ for its set of vertices; X_D for its successor function; $\mathscr{R}(D)$ for its roots (list or set, depending on context); $\mathscr{V}ar(D)$ for the set of variables labeling its sprouts; $\mathscr{MV}ar(D)$ for the multiset of those variables; $\mathscr{A}cc(D) = X^*(R)$ 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.

2.1 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* $\mathscr{D}om(\xi)$ and *image* $\mathscr{I}m(\xi)$ 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. $s \in \mathscr{D}om(\xi_D)$ and $L(s) = L(t)$ imply $t \in \mathscr{D}om(\xi_D)$ and $\xi_D(s) = \xi_D(t)$ for all sprouts $s,t \in S$;
2. $s \in \mathscr{D}om(\xi_{D'})$ and $L'(s) = L'(t)$ imply $t \in \mathscr{D}om(\xi_{D'})$ 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 $\mathscr{V}er(D) \cap \mathscr{V}er(D') = \mathscr{V}ar(D) \cap \mathscr{V}ar(D') = \varnothing$, we define the disjoint union of extensions as $\langle D,\xi\rangle \oplus \langle D',\xi'\rangle = \langle D \oplus D', \xi \cup \xi'\rangle$.

Sprouts labelled by the same variable should be connected to the same vertex, which must then occur several times in the list of roots, 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 $\xi_D(\mathscr{D}om(\xi_D))$ must be a set, making the set difference $[1\,..\,|R'|] \setminus \xi_D(\mathscr{D}om(\xi_D))$ 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 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 $v \in (S \cup S') \setminus \mathscr{D}om(\xi)$, then $\xi^*(s) = v$.
3. If $v \in \mathscr{D}om(\xi)$, 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 1 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 4 (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. $V'' = (V \cup V') \setminus \mathscr{D}om(\xi)$;
2. $S'' = (S \cup S') \setminus \mathscr{D}om(\xi)$;
3. $R'' = \xi^*(R([1\,..\,|R|] \setminus \xi_{D'}(\mathscr{D}om(\xi_{D'})))) \cup \xi^*(R'([1\,..\,|R'|] \setminus \xi_D(\mathscr{D}om(\xi_D))))$;
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 2 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 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 1): $\xi^*([1,2,3] \setminus [2]) = \xi^*([1,3]) = [f,h]$; 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

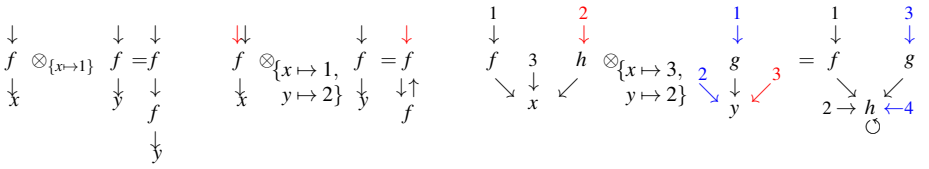


Fig. 1 Different forms of composition: substitution, formation of a cycle, and transfer of roots.

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 [7], that we recall now.

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

Lemma 1 (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 $\langle 1^Y_X, \xi \rangle$. Further $C \otimes_\xi D = D$ iff $X = \mathscr{V}ar(D)$ and α is the identity map.*

Lemma 2 (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 $(U \otimes_\theta V) \otimes_\gamma W = U \otimes_\xi (V \otimes_\zeta W)$.*

Lemma 2 is proved in a particular case in [9]. Here is the proof for the general case that is needed later in the proof of Lemma 3.

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, $\mathscr{D}om(\xi_{U \to W}) \cap \mathscr{D}om(\zeta_{V \to W}) = \mathscr{D}om(\xi_{W \to U}) \cap \mathscr{D}om(\zeta_{W \to V}) = \varnothing$ on the one hand, and $\mathscr{I}m(\xi_{U \to W}) \cap \mathscr{I}m(\zeta_{V \to W}) = \mathscr{I}m(\xi_{W \to U}) \cap \mathscr{I}m(\zeta_{W \to V}) = \varnothing$ on the other hand. The equality then follows straightforwardly. □

2.2 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 right-hand sides of rules have the same number of roots so as to avoid the creation of ill-formed drags.

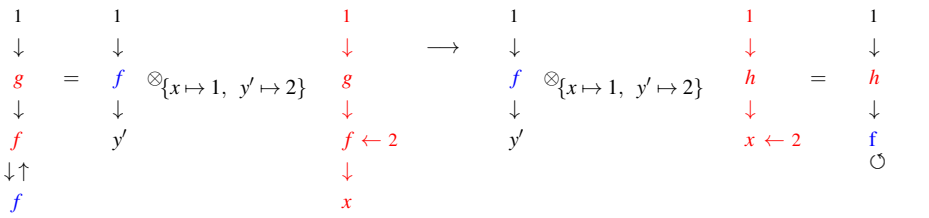


Fig. 2 Rewriting and cycles.

Definition 5 (Rules) A *drag rewrite rule* is a pair of clean drags[1], written $L \to R$, such that (i) $|\mathscr{R}(L)| = |\mathscr{R}(R)|$, and (ii) $\mathscr{V}ar(R) \subseteq \mathscr{V}ar(L)$.

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 6 (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 7 (Rewriting) Let $\mathscr{R}$ be a graph rewrite system. We say that a nonempty clean drag D *rewrites* to a clean drag D', and write $D \longrightarrow_{\mathscr{R}} D'$, iff $D = C \otimes_\xi L$ and $D' = (C \otimes_\xi R)$ for some drag rewrite rule $L \to R \in \mathscr{R}$ 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 3 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.

Example 4 This time, the rooted term $f(a,b)$ is rewritten to the drag made of two components, the rooted terms $g(a)$ and b. Note that allowing the non-clean right-hand side made of the rooted drag $g(x)$ and the non-rooted term y, as in [9], would result in the clean rooted term $g(a)$, the component b being then rootless and thrown away.

We are now finished with the material from [9] needed for the rest of this paper.

[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 $\mathscr{MV}ar(L) = \mathscr{MV}ar(R)$. This was exploited in [9] to simplify some technicalities. We prefer the simpler definition here since, being interested in unification, the rôle of right-hand sides becomes secundary.

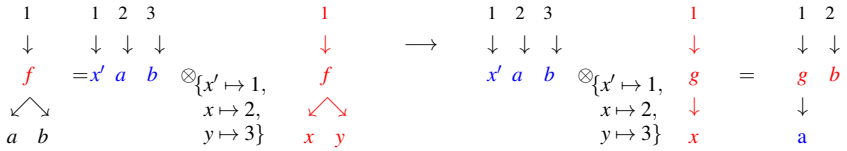


Fig. 3 Rewriting and connected components.

3 Unification

Unification of two terms is somewhat simple: one term is identified with a subterm of the other by some substitution applying to both. As we have seen, a substitution is just a particular case of composition.

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 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, $\mathscr{D}om(\mathscr{R}(U))=[1..m]$ and $\mathscr{D}om(\mathscr{R}(V))=[m+1..m+n]$. 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 8 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 9 Two drags U,V are *identified* with a drag W at *partner vertices* $(\overline{u},\overline{v})$ by an injective function $o:\mathscr{V}er(U)\cup\mathscr{V}er(V)\to\mathscr{V}er(W)$ called *identification* (extended to lists of vertices in the natural way), which we write $U[\overline{u}]=_o V[\overline{v}]$, iff:

1. $\forall w\in\mathscr{V}er(U),w'\in\mathscr{V}er(V)$ such that $o(w)=o(w')$, $w:f$ iff $w':f$ iff $o(w)=o(w'):f$;
2. $\forall w\in\mathscr{V}er(U),w'\in\mathscr{V}er(V)$ 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 $\mathscr{V}er(U)$ and $\mathscr{V}er(V)$).

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 10 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 5 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.

3.1 Subsumption

Definition 11 We say that a drag U is an *instance* of a drag V, or that V *subsumes* U, and write $U \succeq V$, 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 3 *$\succeq$ is a quasi-order, called* subsumption*, whose equivalence is variable renaming and strict part is a well-founded order.*

Proof The relation $\succeq$ being reflexive, we show transitivity. Let U, V, W be three drags whose sprouts are labelled by pairwise disjoint sets of variable, such that $U \succeq V \succeq W$. 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 2, $U = E \otimes_\theta W$, for some context extension $\langle E,\theta\rangle$ of W, hence $U \succeq W$.

Assume that $U \succeq V \succeq U$. 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 1 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 that $U > V$. 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. □

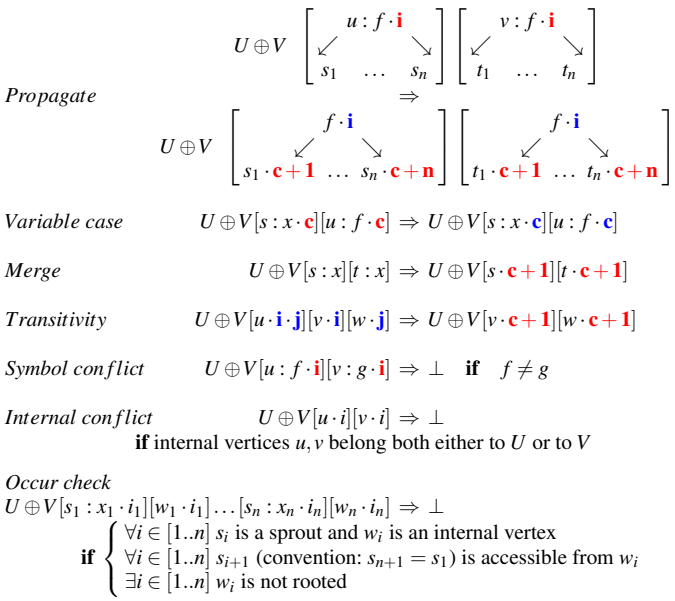


Fig. 4 Drag unification rules

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 [7], 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 well-founded subsumption quasi-order to context extensions:

Definition 12 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 $(D, \zeta) \succeq_U (C, \xi)$, if $(D \otimes_\zeta U)$ is an instance of $(C \otimes_\xi U)$.

3.2 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 hold marks $\mathbf{1}, \ldots |\overline{\mathbf{u}}|$), 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 $\mathbf{i_1}, \ldots, \mathbf{i_n}$. 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 one of the two following situations occurs: two sprouts hold the same variable; vertex marked **i** and **j** provides a link between two vertices marked **i** and **j** respectively. In both cases, such a pair of vertices must now be marked in red if not marked already, giving then 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 [7], 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 4 computes an equivalence relation between the vertices of two drags to be unified from which their most general unifier is extracted in Section 3.4. 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 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 [29]. A related idea appears also in [22].

Example 6 In our example of Figure 5, 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. When a red marl labels a sprout, as in step **4**, the mark is simply turned blue, as explained early on in this paragraph.

Unifying the same drags at roots $r:1$ and $r:3$ would increment all marks by 1.

Example 7 An example of failure is given at Figure 6, with a variable shared between both drags to be unified. At step 4, the right-hand sides successors of the marked pair are already marked, hence need not be marked again. Unification proceeds in two rounds separated by an application of *Transitivity* and gives a failure.

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 4 *Unification rules terminate in quadratic time and space w.r.t. the size of the input.*

Proof Since a pair of identical vertices is never marked, 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 $(|U| + |V|) \times (|U| + |V| - 1)$. □

3.3 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.

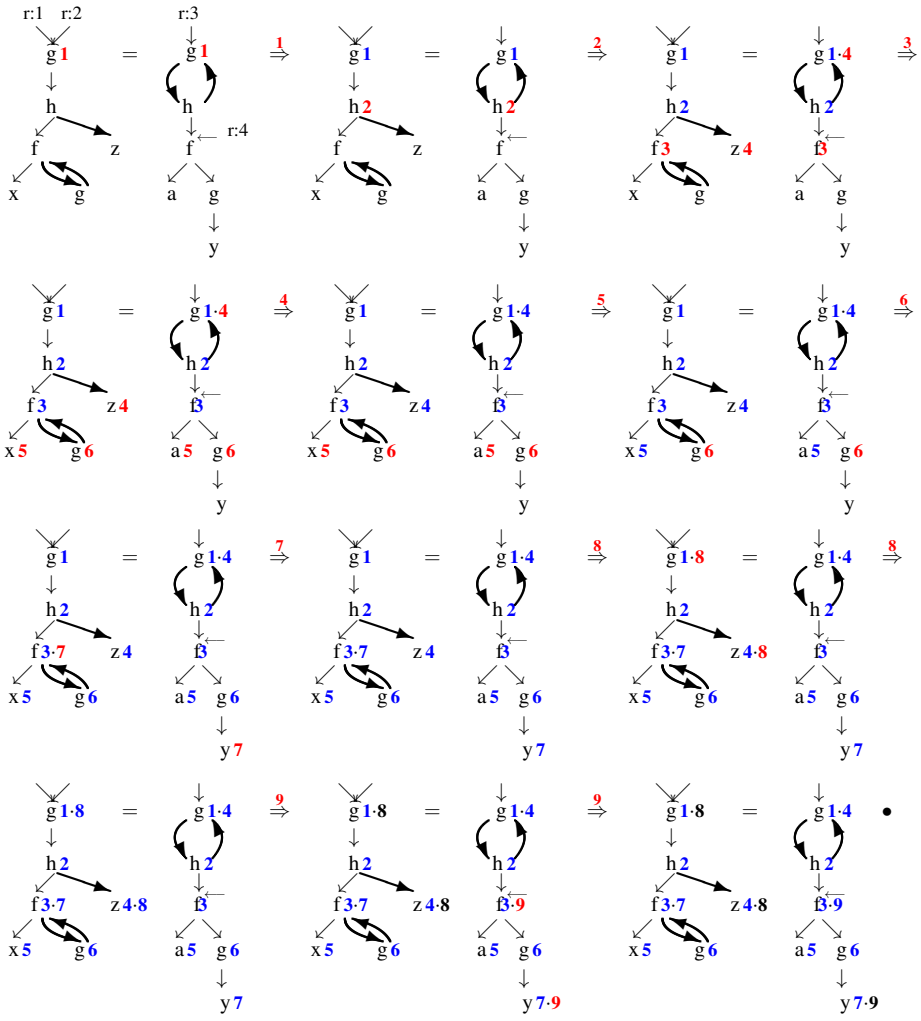


Fig. 5 Successful unification of two patterns.

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

Definition 13 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 $U =_k V$ when omiting the partner vertices $\overline{u}$ and $\overline{v}$, the unification problem obtained at step k of rewriting the initial unification problem $U[\overline{u}] = V[\overline{v}]$, also denoted by $U =_0 V$. When mentionning vertices in $U =_k V$ which may differ

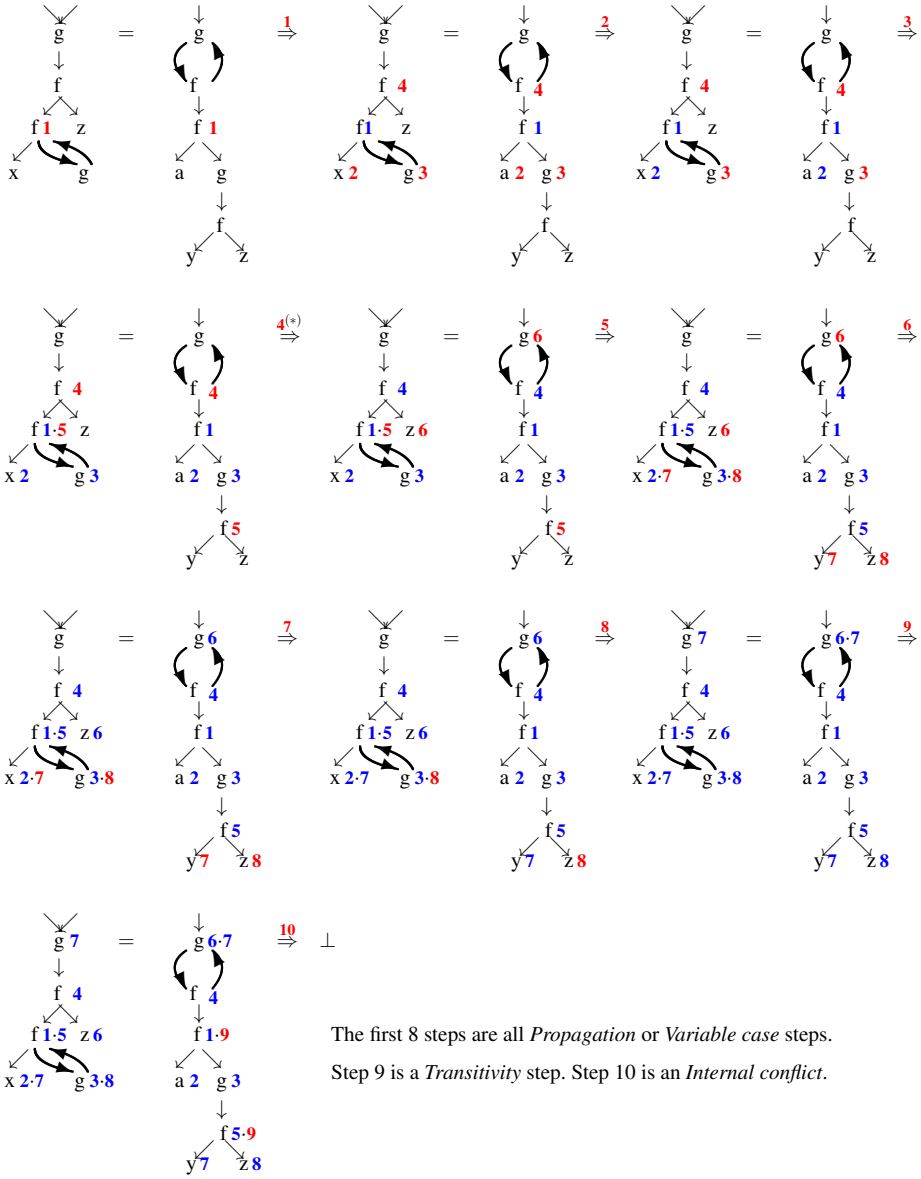


Fig. 6 Unification of two patterns (failure case).

from the partner vertices $\overline{u},\overline{v}$, we shall write instead $U =_k V\,[w_1]\ldots[w_n]$, the vertices $w_1,\ldots w_n$ belonging to either of U,V.

Definition 14 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 5 *$\equiv_k \subseteq \equiv_l$ 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, thanks to the corresponding rule, hence coincides with $\equiv_{unif}$:

Lemma 6 *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 5, $U =_p V[u \cdot j][w \cdot j \cdot i][v \cdot i]$ for some p, hence $U =_q V[u \cdot k][v \cdot k]$ for some k, q by *Transitivity*. □

Lemma 7 *$\equiv_{unif}$ 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 8 *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 7, 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 9 *Let $U = V[s_1 : x_1 \cdot i_1][w_1 \cdot i_1] \ldots [s_n : x_n \cdot i_n][w_n \cdot i_n]$ 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 10 *An internal conflict has no solution.*

Proof By lemma 8, u and v should be identified, but no clean extension can identify two different internal vertices of a drag. □

Lemma 11 *Unification rules are sound.*

Proof Soundness of *Internal conflict* and *Occur check* is shown at Lemmas 10, 9. Soundness of all other rules is a consequence of equality of drags. □

3.4 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 15 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 $s : x \equiv uX_D^+ t : y$ 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 16 (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.

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

Lemma 12 *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 7, $\equiv_{unif}$ 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 6, $U = V[u \cdot i][v \cdot i]$ for some i, hence *Internal Conflict* applies.
3. Assume $\equiv_{unif}$ does not have enough roots. Then, by Lemma 6, *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 17 (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 $u \equiv v$.

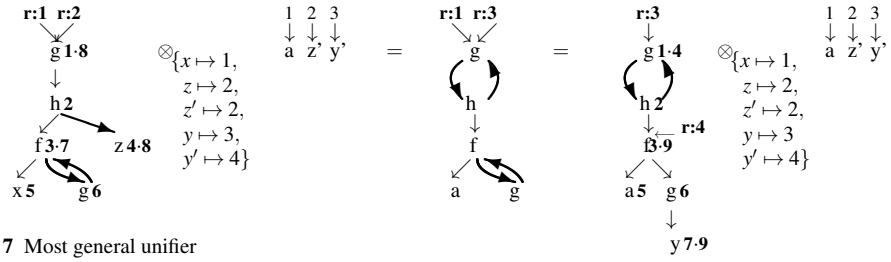


Fig. 7 Most general unifier

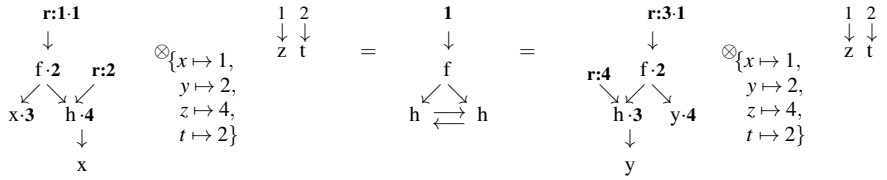
– **induction step:** let M be a non-trivial equivalence class of vertices containing at least one variable. Then, $mgu(U \oplus V, \equiv) = (C, \xi) \oplus mgu((U \oplus V) \otimes_\xi C, (\equiv \setminus M) \cup \{o(M)\})$, where:
  1. if M includes two or more variables, but no internal vertices, $M = \{s_1 : x_1, \ldots s_n : x_n\}$, with $n > 1$, then C consists of just one sprout s, labelled with some fresh x, equipped with n roots $[1 \ldots n]$, $\forall i \in [1..n]\ \xi(s_i) = i$, and $o(s_i) = s : x$.
  2. if $M = \{s_1 : x_1, \ldots s_n : x_n\} \cup N$, 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 $[1 \ldots n]$, $\xi(s_i) = i$, and $o(s_i) = o(v) = o(w) = v$.
  3. if $M = \{s_1 : x_1, \ldots s_n : x_n\} \cup N$, 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 consists of just one sprout s, labelled with some fresh y, equipped with n roots $[1 \ldots n]$, $\xi(x_i) = i, \xi(y) = k$, and $o(x_i) = o(v) = o(w) = v$.

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 8 (Continued) Figure 7 shows the context drag, switchboard, and overlapping drag obtained by composition of the inputs drags of Figure 5. The equivalence on vertices considered here is defined by having equal markings, which is in solved form as shown at Lemma 12. Note that the resulting drag has two roots, not one as might have been expected. This is so because the marking results from unification at partner vertices (f, f), which are then identified by case 2 of Definition 17, but the roots $r : 1$ and $r : 3$ belong to trivial classes, hence are not identified. Unifying at vertices $(r : 1, r : 2)$ instead would give a single root.

Example 9 Figure 8 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 $(r : 1, r : 3)$.

Lemma 13 *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.*



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.

Fig. 8 Building a most general unifier from a solved congruence

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 strictly 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 17, $\langle C, \xi \rangle$ 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 1 *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 13, $\langle C, \xi \rangle = mgu(U[\overline{u}] \oplus V[\overline{v}], \equiv)$ 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. □

3.5 Most general unifiers

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

Theorem 2 *Unification is unitary, and has quadratic worst case time and space complexity.*

Proof The first part of the claim follows from Lemma 11 and Theorem 1. For the second part, Lemma 4 shows that a solved form is obtained in quadratic time and space. Further, the solved form itself has a linear size in terms of the input unification problem, since the algorithm never generates new terms. As the construction of the most general unifier from the solved form is clearly done in linear time in terms of its size, hence in terms of the size of the input problem, the whole process takes at most quadratic time and space. □

4 Confluence

Confluence of a terminating term rewriting system follows from the joinability of its critical pairs, obtained by unifying overlapping left-hand sides of rules [23]. Our goal is to generalize this result to the drag framework.

To simplify the development, we assume here that left-hand sides of rules are connected drags. The general case can be carried out to the price of more involved statements and technicalities. We leave it as an exercise for the reader.

Lemma 14 *Let $S \longleftarrow_{L\to R} U \longrightarrow_{G\to D} T$, and assume that U has no internal vertex being at the same time an internal vertex of L and of G. Then, there exist two drags V, W and a switchboard ξ such that $U = V \otimes_\xi W$, $V \longrightarrow_{L\to R} V'$, $W \longrightarrow_{G\to D} W'$, $S = V' \otimes_\xi W$ and $T = V \otimes_\xi W'$.*

Proof By definition of rewriting, there exist rewriting extensions $\langle A, \xi\rangle$ and $\langle B, \zeta\rangle$ such that $U = A \otimes_\xi L = B \otimes_\zeta G$. We assume without loss of generality that L and G are renamed apart as well as A and B, making $\xi_L \cup \zeta_G$ well defined, as well $\mathscr{R}(A) \cup \mathscr{R}(B)$.

Let now C be the drag whose internal vertices are those of U which are not internal vertices of either L or G, its roots and sprouts are those of A plus those of B, and its successor relationship is inherited from that of U.

Since L and G do not share internal vertices by assumption, $B = L \otimes_\xi C$ while $A = G \otimes_\zeta C$, hence $U = L \otimes_\xi C \otimes_\zeta G$ (using the strong form of associativity given in [9] –although the weak form given here would suffice). Take $V = L$ and $W = C \otimes_\zeta G$. □

Lemma 15 (Commutation) *Assume that $U = V \otimes_\xi W$, $V \longrightarrow_{L\to R} V'$ and $W \longrightarrow_{G\to D} W'$. Then, $U \longrightarrow_{L\to R} V' \otimes_\xi W \longrightarrow_{G\to D} V' \otimes_\xi W'$ and $U \longrightarrow_{G\to D} V \otimes_\xi W' \longrightarrow_{L\to R} V' \otimes_\xi W'$.*

Proof Easy consequence of associativity of composition. □

Lemma 16 (Critical Pair Lemma) *Let $S \longleftarrow_{L\to R} U \longrightarrow_{G\to D} T$, and let us assume that U has an internal vertex w which is an internal vertex of both L and G. Then, there exist $\overline{u} \in \mathscr{V}er(L)$ and $\overline{v} \in \mathscr{V}er(G)$), and a unifying extension $\langle D, \zeta\rangle$ of the equation $L[\overline{u}] = G[\overline{v}]$ such that $U = L \otimes_\zeta D = G \otimes_\zeta D$.*

Proof Let A be the subset of internal vertices of U which are also internal vertices of L and of G, roots of L, U, G being considered as specific internal vertices. By assumption, $A \neq \varnothing$, implying that L and G overlap. The core of the proof is the definition of two lists $\overline{v}, \overline{w}$ of partner vertices of L, G which generate A, that is, all vertices of A are accessible from $\overline{v}$ in L and from $\overline{w}$ in G. As partner vertices, v_i and w_i must coincide, that is, be identified in A.
Let us denote vertices of A by u, v and w according to their origin, in U, L and G respectively.

Because left-hand sides of rules are clean drags, for all internal vertices $u \in A$, there exists some root $r \in \mathscr{R}(L) \cup \mathscr{R}(G)$ such that, $r\,(X_L^* \cup X_G^*)\,v$. There are therefore three kinds of partner vertices (v, w): v, w are roots of both V and W; or v is a root of V and w is not a root of W; or w is a root of w and v is not a root of V. Eliminating redundancies (with respect to accessibility) yields two lists of vertices that satisfy the conditions for being partner vertices, and generate A.

We now show that U defines a unifying extension of the equation $L[\overline{v}] = G[\overline{w}]$. Since L and G match U, $U = L[\overline{v}] \otimes_\gamma C = G[\overline{w}] \otimes_\delta D$ for some rewriting extensions $\langle C, \gamma\rangle$ and $\langle D, \delta\rangle$. Assuming that L, G are renamed apart, then $U = (L[\overline{v}] \oplus G[\overline{w}]) \otimes_{\gamma\cup\delta} (C \oplus D)$. Hence L and G are unifiable at partner vertices $(\overline{v}, \overline{w})$. □

Definition 18 (Critical pair) Let $L \to R$ and $G \to D$ be two rules that are unifiable at partner vertices $\overline{v}, \overline{w}$, and $\langle C, \xi\rangle$ be an mgu. Then, $\langle L \otimes_\xi C, G \otimes_\xi C\rangle$ is called a *critical pair* of $L \to R, G \to D$ at $\overline{v}, \overline{w}$.

Given a drag rewriting system, how many critical pairs can be generated ? Their number is indeed bounded by the potential choices for partner vertices, which must satisfy the constraints stated in the proof of Lemma 16, and resist the inequality test in *Symbol conflict*. In practice, this number should remain small, as is the case for terms.

We can now end up with our second main result:

Theorem 3 *Let $\mathscr{R}$ be a rewrite system on drags. Then, $\mathscr{R}$ is locally confluent iff all its critical pairs are joinable.*

Proof Let $S \longleftarrow_{L\to R} U \longrightarrow_{G\to D} T$, L and G being renamed apart. There are two cases:

1. G has no vertex in common with L. We then conclude by Lemmas 14 and 15.
2. G has a vertex in common with L. By Lemma 16, there is a critical pair. By property of most general unifiers, the joinability of the critical pair can be lifted to S and T. □

As a direct consequence, a terminating drag rewrite system is confluent iff its critical pairs are joinable. If the rewrite system is non-terminating, confluence can still be achieved by using Van Oostrom's decreasing diagrams technique. Developing confluence criteria along these lines goes beyond the objectives of this paper. The case of terms has been thoroughly investigated, see [15, 16, 26]. We believe that similar investigations can be carried out for drags.

A particular case worth mentioning, as suggested to us by Jan-Willem Klop, is orthogonality. Orthogonal term rewriting systems are confluent, whether terminating or not. It so happens with drag rewriting systems, with the exact same definition of orthogonality:

Definition 19 A drag rewriting system is said to be *orthogonal* if it is critical pair free and all its rules are left-linear.

Theorem 4 *Orthogonal drag rewriting systems are confluent.*

Proof The same proof as for terms goes through: parallel rewrites are indeed strongly confluent, which ensures confluence of drag rewriting. Details are left to the reader.

5 Related work

The first, dominant model for graph rewriting was introduced in the mid-seventies by Hartmut Ehrig and his collaborators [14]. 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 [33]. In particular, termination 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 [6] for an old detailed account of these techniques, and [19] for a survey of implementations of various forms of graph rewriting and of available analysis tools.

DPO applies to any category of graphs that has pushouts and unique pushout complements [12]. 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 [12], arbitrary adhesive graph categories [12], single pushout transformation (SPO [13, 34]), or Sesqui-Pushout transformation (SqPO [5]), AGREE [4], and *Hyperedge Replacement Systems* [11]. A detailed comparison of the approach based on drags with all these approaches is not obvious and will be carried out in another paper [10].

DRAG rewriting aims instead 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 DRAG transformation is *port graph rewriting* [17], 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 [2].

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., [29]). A preliminary attempt to handling variables in graph unification is [28], where variables are used to represent labels equipping the vertices or edges. A quite more general approach is [33], 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 [21]. Graphs in GP2 are symbolic graphs whose attributes's values are given by variables satisfying some set of constraints [27]. Variables are not substituted by graphs, but by constrained values.

In contrast, drag variables are real variables as in terms, and drag unification is shown here unitary, and decidable in quadratic time and space, a bound which we believe is not tight. 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. The complexity analysis exploits the fact that the successors of a vertex are ordered and their number is fixed by the symbol labelling that vertex. Relaxing these constraints would blow up the number of most general unifiers resulting in a non-polynomial complexity of matching and unification.

Confluence of graph transformation systems was first studied by Plump, who defined the notion of graph critical pairs and proved their completeness, but also showed that local confluence is undecidable already for terminating systems [30–32]. He also considered the case of symbolic graphs [20]. A main problem with Plump's notion of critical pair is that, to compute the set of all critical pairs, we need a procedure that is exponential on the size of the lefthand sides of the rules. More precisely, according to Plump's definition, the set of critical pairs of two rules r_1, r_2 consists of all pairs of transformations $H_1 \longleftarrow_{r_1} G \longrightarrow_{r_2} H_2$ that are *parallel independent* (e.g., see [12]) and such that G is an overlap of L_1 and L_2. This means that, in principle, to compute all possible critical pairs, we need to compute all possible overlaps of L_1 and and L_2 and to check if they are parallel independent. Moreover, even if it is difficult to estimate what is the exact number of critical pairs, since it is difficult to estimate how many of these pairs of transformations will be parallel independent, we know that many of them are useless. Consequently, less prolific notions of critical pairs are introduced in [1, 24, 25]. For instance, [24] includes an example to show how large may be the different number of critical pairs depending on the approach considered. The example considers the definition of finite automata in terms of graph transformation. More precisely, a finite automaton is represented by a graph including: a) the state/transition diagram of the automaton b) a cursor (represented by a loop) on the vertex denoting the current state of the automaton, and c) a queue of symbols representing the word to be recognized. Then, the transformation rules describe how the given automaton works, i.e., when the first symbol in the queue is recognized by the automaton, the movement of the cursor and the deletion of the symbol. In this example, computing the critical pairs of that rule with itself gave the following results: the number of all the overlaps of the lefthand side of the rule with itself was 51,602; the number of critical pairs according to Plumps definition was 21,478; the number of critical pairs computed using the method presented in [25] was 49; finally, the number of critical pairs computed using the method presented in [1, 24] was 7.

Recently, local confluence was shown decidable for terminating graph rewriting with interfaces [2], where an interface is a subset of the indices of the given graph that are used to define an operation of graph composition by connecting the interfaces of the given graphs. Then, rewriting a graph with an interface, according to [2], means rewriting the graph but leaving the interface invariant. For instance, it would not be allowed to apply a rule if, as a consequence, a vertex in the interface would be deleted or if two vertices in the interface would be merged. With respect to confluence, a main difference between standard DPO rewriting and this variation is that, in DPO rewriting, two graphs G_1, G_2 are considered joinable if they can be rewritten into isomorphic graphs H_1, H_2, respectively, but when the graphs have an interface I it would be required the existence of an isomorphism $h : H_1 \to H_2$ such that $h(v) = v$, for every $v \in I$. This difference is the reason why joinability of critical pairs in standard DPO graph transformation does not imply local confluence, while

that implication holds for graphs with interfaces, implying the decidability of confluence of terminating DPO rewriting of graphs with interfaces. Let us see an example from [2]:

Consider the following rewrite rules on undirected graphs with labelled edges, where $\leftarrow [a] \rightarrow$ represents an edge labelled by a and vertices are subindexed with numbers 1 and 2 to identify them and make the morphisms explicit:

$$r_1\colon \quad [\bullet_1 \xrightarrow{a} \bullet_2] \longleftarrow [\bullet_1 \quad \bullet_2] \longrightarrow [\bullet_1 \circlearrowright \quad \bullet_2]$$

$$r_1\colon \quad [\bullet_1 \xrightarrow{a} \bullet_2] \longleftarrow [\bullet_1 \quad \bullet_2] \longrightarrow [\bullet_1 \quad \bullet_2 \circlearrowright]$$

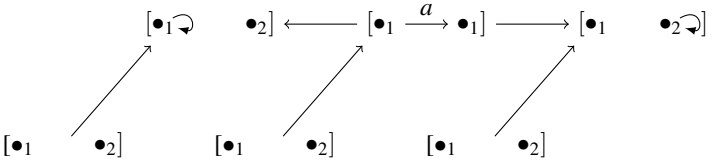
Then, among the possible critical pairs only the following two have non-trivial overlap:

$$[\bullet \circlearrowright \quad \bullet] \longleftarrow [\bullet \xrightarrow{a} \bullet] \longrightarrow [\bullet \quad \bullet \circlearrowright] \qquad \text{and} \qquad [\bullet \circlearrowright] \longleftarrow [\bullet \circlearrowright] \longrightarrow [\bullet \circlearrowright]$$

which are trivially joinable. However, the two rules are not confluent, as we can see below:

$$[\bullet \circlearrowright \xrightarrow[b]{} \bullet] \longleftarrow [\bullet \underset{b}{\overset{a}{\rightleftarrows}} \bullet] \longrightarrow [\bullet \xrightarrow[b]{} \bullet \circlearrowright]$$

Let us see what happens when we work with graphs with interfaces. If we associate an interface, consisting of the two nodes 1 and 2, to the first graph that gave rise to the first critical pair above, we have:



but $([\bullet_1 \quad \bullet_2] \longrightarrow [\bullet_1 \circlearrowright \quad \bullet_2])$ and $([\bullet_1 \quad \bullet_2] \longrightarrow [\bullet_1 \quad \bullet_2 \circlearrowright])$ are not isomorphic anymore, which means that this critical pair is not joinable, witnessing that the two rules are not confluent when applied to graphs with interfaces.

The authors point out that the situation is similar to the term rewriting case, for which local confluence of terminating systems is undecidable for ground rewriting rules, but decidable for open ones. This is only partially true, because in this case the interfaces do not play the rôle of variables, since the interfaces are not in the rules, but in the graphs that we rewrite, while in term rewriting the variables are in the rules (even if the terms that are rewritten may also have variables). When rewriting graphs with interfaces, the interfaces restrict the application of rules, since the interfaces must be preserved, while this is not the case when rewriting terms with rules including variables.

In the case of drags, the analogy to term rewriting is more faithful: confluence of graph/drag rewriting is not decidable for ground rules, but we have shown that it is decidable for rules with variables. Furthermore, as with term rewriting, drag critical pairs are computed via most general unifiers: all of them are usefull in the absence of critical pair criteria. The reason why their number is limited, at most quadratic in the size of the input left-hand sides, is that, in our model, two vertices labeled by the same function symbol have the same number of successors, and these successors are ordered. Allowing for a variable number of successors (via associative function symbols) and disregarding their order (via commutative function symbols), as in Plump's model, would blow up the number of most general unifiers, hence of critical pairs. However, we would only need checking the most general ones, as is the case with term rewriting.

6 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 [9]. 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 [22]. 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. Guilherme Grochau Azzi, Andrea Corradini, and Leila Ribeiro. On the essence and initiality of conflicts in m-adhesive transformation systems. *J. Log. Algebr. Meth. Program.*, 109, 2019.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. Nachum Dershowitz and Jean-Pierre Jouannaud. Drags: A compositional algebraic framework for graph rewriting. *Theor. Comput. Sci.*, 777:204–231, 2019.
10. Nachum Dershowitz, Jean-Pierre Jouannaud, and Fernando Orejas. Graph rewriting cum drag rewriting. Work in progress.

11. Frank Drewes, Hans-Jörg Kreowski, and Annegret Habel. Hyperedge replacement, graph grammars. In Rozenberg [35], pages 95–162.
12. Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of algebraic graph transformation*. Springer-Verlag, 2006.
13. 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 [35], pages 247–312.
14. 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.
15. Bertram Felgenhauer. Rule labeling for confluence of left-linear term rewrite systems. In *IWC*, pages 23–27, 2013.
16. Bertram Felgenhauer, Aart Middeldorp, Harald Zankl, and Vincent van Oostrom. Layer systems for proving confluence. *ACM Trans. Comput. Log.*, 16(2):14:1–14:32, 2015.
17. 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.
18. Annegret Habel, Hans-Jörg Kreowski, and Detlef Plump. Jungle evaluation. *Fundam. Inform.*, 15(1):37–60, 1991.
19. 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.
20. Ivaylo Hristakiev and Detlef Plump. Checking graph programs for confluence. In Martina Seidl and Steffen Zschaler, editors, *Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops, Marburg, Germany, July 17-21, 2017, Revised Selected Papers*, volume 10748 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2017.
21. Ivaylo Hristakiev and Detlef Plump. A unification algorithm for GP 2 (long version). *CoRR*, abs/1705.02171, 2017.
22. Gérard Huet. *Unification dans les langages d'ordre 1, ..., ω*. PhD thesis, Université Paris 7, Paris, France, 1976.
23. Donald Knuth and Peter Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
24. Leen Lambers, Kristopher Born, Fernando Orejas, Daniel Strüber, and Gabriele Taentzer. Initial conflicts and dependencies: Critical pairs revisited. In *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig*, pages 105–123, 2018.
25. Leen Lambers, Hartmut Ehrig, and Fernando Orejas. Efficient conflict detection in graph transformation systems by essential critical pairs. *Electr. Notes Theor. Comput. Sci.*, 211:17–26, 2008.
26. Jiaxiang Liu, Jean-Pierre Jouannaud, and Mizuhito Ogawa. Confluence of layered rewrite systems. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 423–440. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
27. Fernando Orejas and Leen Lambers. Symbolic attributed graphs for attributed graph transformation. *ECEASST*, 30, 2010.
28. 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.
29. Mike Paterson and Mark N. Wegman. Linear unification. *J. Comput. Syst. Sci.*, 16(2):158–167, 1978.
30. Detlef Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In Ronan Sleep, Rinus Plasmeijer, and Marko van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, pages 201–213. John Wiley, 1993.
31. Detlef Plump. Critical pairs in term graph rewriting. In Igor Prívara, Branislav Rovan, and Peter Ruzicka, editors, *Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science (MFCS '94), Kosice, Slovakia*, volume 841 of *Lecture Notes in Computer Science*, pages 556–566. Springer, August 1994.
32. Detlef Plump. Confluence of graph transformation revisited. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 280–308. Springer, 2005.
33. 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.
34. Jean-Claude Raoult. On graph rewritings. *Theor. Comput. Sci.*, 32:1–24, 1984.
35. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.