Unification of Drags and Confluence of Drag Rewriting

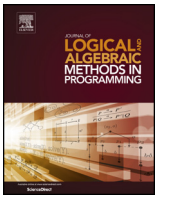
Jean-Pierre Jouannaud, Fernando Orejas

Journal of Logical and Algebraic Methods in Programming 131 (2023) 100845

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



Unification of drags and confluence of drag rewriting ☆

Jean-Pierre Jouannaud [a], Fernando Orejas [b]

[a] Université de Paris-Saclay, Projet INRIA Deducteam, Laboratoire de Méthodes Formelles, Saclay, France
[b] Universitat Politècnica de Catalunya, Department of Computer Science, Barcelona, Spain

ARTICLE INFO

Article history:
Received 11 August 2022
Received in revised form 18 November 2022
Accepted 8 December 2022
Available online 22 December 2022

Keywords:
Drags
Graphs
Unification
Confluence

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.

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, [4].

Our work is based on a recent, purely combinatorial, view of labeled graphs [9]. Drags are labeled graphs equipped with roots and *sprouts*, which are vertices without successors labeled 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 indeed the rôle of both context grafting and substitution in the case of terms.

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. Propagation may of course fail, for example at a pair of vertices labeled by different function symbols. When it succeeds, a most general unifier can be extracted from the propagation's result.

☆ This work is partially supported by MCIN/AEI /10.13039/501100011033 under grant PID2020-112581GB-C21.
E-mail address: jeanpierre.jouannaud@gmail.com (J.-P. Jouannaud).

https://doi.org/10.1016/j.jlamp.2022.100845

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-overlapping 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 are 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. We use $\varnothing$ for an empty list, set, or multiset, $\cup$ for set and multiset union, $\cdot$ 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. $[1..n]$ is the set (or list) of natural numbers from 1 to n. 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. We presuppose: a set of nullary variable symbols Ξ, used to label some vertices without outgoing edges, called *sprouts*; and a set of function symbols Σ, disjoint from Ξ, whose elements, equipped with a fixed arity, are used as labels for all other vertices called *internal*.

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. $R : [1 .. |R|] \to V$ is a finite list of vertices, called *roots*;
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.

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

We use R for both the function itself of *domain* $\mathcal{D}om(R) = [1..n]$ and its resulting list $[R(1) .. R(n)]$ of length $|R| = n$.
The labeling function extends elementwise to lists, sets, and multisets of vertices.
We introduce below some classical vocabulary, mostly originating from graph theory.

Definition 2. Let $D = \langle V, R, L, X, S\rangle$ be a drag. If $v \in X(u)$, then (u, v) is a directed *edge* with *source* u and *target* v. We also write uXv. We also say that v is a *successor* of u, and u a *predecessor* of v. The reflexive-transitive closure X^* of the relation X is called *accessibility*. A vertex v is said to be *accessible* (or *reachable*) from vertex u, and likewise that u *accesses* v, if uX^*v, otherwise it is *unreachable* from u. u is a *true ancestor* of v if v is reachable from u but u is unreachable from v. Vertex v is *accessible* (or *reachable*) if it is accessible from some root, and *unreachable* otherwise. A sprout is *isolated* if it has no predecessor. A vertex u is *rooted* if it occurs, possibly several times, in the list R, otherwise it is *rootless*. A drag is *clean* if all its vertices are accessible, and *linear* if no two sprouts have the same label.

It will be convenient, in particular in examples, such as Examples 1 and 2, and pictures 1, 2, and 6, to identify a sprout of a linear drag with the variable symbol that is its label. If a drag is non-linear, with n sprouts sharing the same variable label x, we will then denote these sprouts by $x_1, x_2, \ldots x_n$. Similarly, the label of a vertex will be used as its name when non-ambiguous. Upper indices will be used to denote roots, the notation $u^{n,m}$ telling us that u appears at positions n and m in the list R. Any vertex, including sprouts, can be rooted, possibly several times. This is essential for having a nice algebra of drags. A further convention is that the n successors of an internal vertex whose label has arity n are drawn from left to right. An internal vertex labeled by a constant (of arity 0) has of course no successors, this is the same for sprouts.

Terms as ordered trees, sequences of terms (forests), terms with shared subterms (dags) and sequences of dags (jungles [18]) are all particular kinds of drags, which are clean when rooted. The drag having no vertex, called the *empty* drag (which is also the empty tree), is clean too.

Example 1. Four drags are depicted in Fig. 1. The leftmost one represents a term equipped with roots, namely $f^{2,3}(x^1, x^4)$, made of one internal vertex labeled f, two sprouts labeled x, and the list of roots (x_1, f, f, x_2). In the graphical representation, roots become arrows going from distinct integers in the interval $[1, N]$ to each rooted vertex, where N is the number of roots in the drag. Notice also that we are implicitly assuming that the arity of f is 2.

The second drag, called D is another term, while the third drag, D', is a dag, which has no roots. Finally, D'' is a drag including two loops.

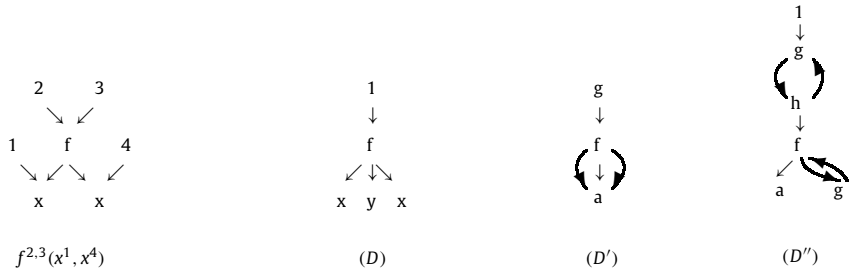


Fig. 1. Four drags.

Definition 3. 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; $\mathcal{I}nt(D)$ for its set of internal vertices; $\mathcal{S}(D)$ for its set of sprouts; X_D for its successor function; $\mathcal{R}(D)$ for its roots (list or set, depending on context); $\mathcal{L}_D$ for its labeling function; $s : x$ if sprout s has variable x for label, $\mathcal{V}ar(D)$ for the set of variables labeling its sprouts; $|D|$, its *size*, for the number of accessible internal vertices plus the size of R; and $in(v, D)$, the *in-degree* of vertex v, for the number of predecessors of v in D plus the number of roots of v in D.

2.1. Equality of drags

Drags are particular graphs, the name of their vertices is not relevant. The order of roots in their list is not relevant either, as we shall see when defining rewriting. What matters is whether two sprouts are labeled by the same or different variables.

Equality of drags will of course play a key role when it comes to unification.

We define a *vertex renaming* to be a bijection between two finite sets of vertices that restricts to internal vertices and sprouts, and a *variable renaming* to be a bijection between two finite sets of variables.

Definition 4. Two drags $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$ in this order, are *equal modulo renaming*, namely a vertex renaming $\iota : V \to V'$, a variable renaming $\alpha : \mathcal{V}ar(D) \to \mathcal{V}ar(D')$ and a permutation σ of $[1\,..\,n]$ (we also say that D' is a *renaming* of D), iff:

1. $\forall u \in V, \mathbf{v} \in V^* :\ X(u) = \mathbf{v}$ iff $X'(\iota(u)) = \iota(\mathbf{v})$
   (extending ι to lists of vertices in the natural way)
2. $\forall u \in \mathcal{I}nt(D) :\ L'(\iota(u)) = L(u)$
3. $\forall s \in S :\ L'(\iota(s)) = \alpha(L(s))$
4. $|R| = |R'| = n$ and $\forall i \in [1\,..\,n] :\ R'(\sigma(i)) = R(i)$.

We then write $D =^{\iota}_{\alpha,\sigma} D'$. The drags D and D' are said to be *equal* modulo variable renaming if ι is the identity, and *identical* if α is the identity as well.

The subscripts α, σ and superscript ι are usually omitted when equal to an identity. They may also be omitted when no ambiguity arises with definitional equality (which actually corresponds to identity with identical lists of roots). In particular, in the absence of annotations, equality should always be interpreted as definitional in definitions.

Equality of drags modulo renaming is an equivalence relation, since the identity is a bijection, inverse of a bijection is a bijection, and bijections compose.

Two drags are *disjoint* if they have no vertex nor variable in common. By *renaming apart* two drags D, E, we mean defining two disjoint drags D', E' such that D' and E' are equal modulo renaming to D and E respectively.

Definition 5. The disjoint union of two drags D, E, written $D \oplus E$ is a drag obtained by first renaming D and E apart, and then forming the union of their labeled vertices and edges, and the concatenation of their roots, those of D coming first. In case D and E don't share vertices and/or variables, their vertices and/or variables will be kept identical so as to facilitate technicalities: $D \oplus E$ will then be the juxtaposition of D and E (in this order). Since juxtaposition is clearly associative (up to vertex renaming), we denote by $\Sigma_i D_i$ the juxtaposition of several drags.

Definition 6. Given drags $D = \langle V, R, L, X, S\rangle$ and $D' = \langle V', R', L', X', S'\rangle$, whose respective internal vertices are I and I', a *(drag) morphism* $o : D \to D'$ is a map from V to V' such that:

1. o restricts to internal vertices: $o(I) \subseteq I'$;
2. o preserves labels of internal vertices: $\forall u \in I :\ L'(o(u)) = L(u)$;

3. o preserves the successor function: $\forall u \in I : X'(o(u)) = o(X(u))$;
4. o forces sharing: $\forall s : x, t : x \in S : o(s) = o(t)$.

Definition 7. A morphism $o : D \to D'$ is an *injection* if

(i) its restriction to I is injective,

(ii) A vertex $v \in I$ must be rooted if there exists an edge $(u', o(v))$ in D', called a *new edge*, such that either $u' \in I' \setminus \mathcal{I}m(o)$, or $u' = o(u)$ for some vertex u of D and (u, v) is not an edge of D.

The notion of injection, injective on internal vertices only, is specific to drags which have variables: different sprouts sharing the same variable label must be mapped to the same vertex, and that vertex can even be the image of some (unique) internal vertex. Property (ii) implies that D' has three kinds of edges: those between internal vertices of D, those between vertices which are not the image of vertices in D, and those which are the image of roots in D. This property is directly related to the definition of composition to come later.

Example 2. Let $D = f(x, y, x)$ and D' be the middle two drags of Fig. 1). The map $o(f) = f$ and $o(x_1) = o(y) = o(x_2) = f$ is an injection from D to D'. Note that root 1 in D is mapped to the edge (g, f) in D' (this is the only possibility here, but there would be others if f were also rooted in D'). Let now D'' be like D', except that it has no sprout and three edges (f, f). The injection from D to D'' is now $o(x_1) = o(y) = o(x_2) = o(f) = f$.

Morphisms ignore names. If o is a drag morphism from C to D and C' is a renaming of C, then composing this renaming with o yields a morphism from C' to D. This remark will be used without saying later, o being then an injection.

As expected, morphisms and injections are closed under composition.

2.2. Composition of drags

In this section we introduce a main operation on drags that generalizes the notion of substitution for trees.

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 8 *(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', equivalently an *extension* $\langle D', \xi\rangle$ of D, is a pair of partial injective functions $\langle \xi_D : S \to \mathcal{D}om(R');\ \xi_{D'} : S' \to \mathcal{D}om(R)\rangle$ such that

1. *S-compatibility*: $\forall s, t \in S$ s.t. $L(s) = L(t) : s \in \mathcal{D}om(\xi_D)$ iff $t \in \mathcal{D}om(\xi_D)$ and $R'(\xi_D(s)) = R'(\xi_D(t))$;
2. *S′-compatibility*: $\forall s, t \in S'$ s.t. $L'(s) = L'(t) : s \in \mathcal{D}om(\xi_{D'})$ iff $t \in \mathcal{D}om(\xi_{D'})$ and $R(\xi_{D'}(s)) = R(\xi_{D'}(t))$;
3. *well-behavedness*: it induces no cycle among sprouts, using ξ, R, R' relationally:
   $\nexists\, n > 0, s_1, \ldots, s_{n+1} \in S, t_1, \ldots, t_n \in S' : s_1 = s_{n+1}, \forall i \in [1\,..\,n] : s_i \xi_D R' t_i \xi_{D'} R s_{i+1}$.

A *switchboard component* ξ_D or $\xi_{D'}$ is *linear* if any two sprouts in its domain have different labels. A switchboard is *linear* in both its switchboard components are. A *rewriting switchboard* for D, D' is a switchboard ξ such that ξ_D is linear and surjective and $\xi_{D'}$ is total. Rewriting switchboards yield *rewriting extensions*.

Three examples of well-behaved switchboards are given in Fig. 2: $\{x \mapsto 1\}$ for the first, $\{x \mapsto 1, y \mapsto 2\}$ for the second, and $\{x \mapsto 3, y \mapsto 2\}$ for the third.

Sprouts labeled by the same variable should be connected by ξ to the same vertex –unless ξ is undefined for them all– which must then occur several times in the list of roots, as required by the first two conditions and the injectivity of switchboard components. These two conditions are of course automatically satisfied by linear switchboards.

Note that we could define ξ as a partial function from $\mathcal{D}om(\xi_D) \cup \mathcal{D}om(\xi_{D'})$ when these domains are disjoint sets, we have actually implicitly used this property in the above explanation and will use it in the sequel whenever convenient. Defining its value would however require us to consider ξ as a pair of functions using R and R' respectively.

Rewriting extensions play a key rôle for defining rewriting later, in which case D' will stand for a left-hand side of rule and D for its context. The conditions mean that all sprouts and roots of the left-hand side of rule must disappear in the composition.

We now move to the composition operation on drags induced by a switchboard. The essence of this operation is that the 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, thanks to well-behavedness, a vertex of one of the two drags must be reached which is not a sprout in the domain of the switchboard. That vertex is called *target*:

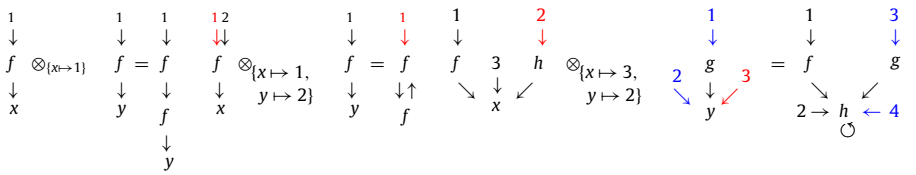


Fig. 2. Different forms of composition: substitution, formation of a cycle, and transfer of roots. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Definition 9 *(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* $\xi^*()$ 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 3. Consider the last of the three examples in Fig. 2, 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 10 *(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. $R'' = \xi^*(R([1\,..\,|R|] \setminus \mathcal{I}m(\xi_{D'})) \cdot \xi^*(R'([1\,..\,|R'|] \setminus \mathcal{I}m(\xi_D))$;
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, then *all* sprouts of D' disappear in the composed drag, while all vertices of D' which are also roots become rootless vertices in the composed drag.

Example 4. We show in Fig. 2 three examples of compositions. The first is a substitution of terms. The second 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 labeled 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 3): $\xi^*([1, 2, 3] \setminus [2]) = \xi^*([1, 3]) = [f, h]$; and $\xi^*([1, 2, 3] \setminus [3]) = \xi^*([1, 2]) = [g, h]$, hence the list of roots of the resulting drag is $[f, h, g, h]$.

The third computation illustrates the fact that composition impacts the list of roots in complex ways. Here, the second root of the left drag (in the composition) pointing at vertex x becomes the second root in the result, pointing now at vertex h, while vertex x has become vertex h. Likewise, the first root of the right drag has become the third root of the result, both pointing at vertex g, and the third, pointing at vertex y, has become the fourth, pointing now at h.

The definition of composition does not assume any property of the input drags. Composing a single-rooted clean drag D having at least one internal vertex with a non-clean drag C consisting of a single non-rooted sprout labeled x, has an observable effect on D: the result of the composition $C \otimes_{\{x\mapsto 1\}} D$ is the drag D', which is D deprived of its root, hence is non-clean since D has internal vertices. In other words, any clean drag D can be sent by an appropriate composition to a drag whose set of accessible vertices is empty. This also implies that D can be sent to any drag U, once cleaned, by taking $C = x \oplus U$.

We conclude this section by showing that drag equality is *observational*:

Lemma 1. *Let D, E be drags that are equal modulo renaming, and $\langle C, \xi\rangle$ an extension of D. Then, there exists an extension $\langle C, \zeta\rangle$ of E such that $C \otimes_\xi D$ and $C \otimes_\zeta E$ are equal modulo renaming.*

Proof. Let $D =^{\iota}_{\alpha,\sigma} E$. It suffices to define $\zeta_C = \sigma \circ \xi_C$ and $\zeta_E = \xi_D \circ \iota^{-1}$, and to extend the bijections ι and σ as the respective identities on the vertices of C which do not belong to $\mathcal{D}om(\xi_C)$, and on the roots of C which do not belong to $\mathcal{I}m(\xi_C)$. □

The important observation is that σ becomes the identity in case ξ_C is surjective, hence explaining why the order of roots in drags is irrelevant as far as rewriting is concerned.

2.3. Drag algebra

Composition has important algebraic properties, existence of identities and associativity [7]. We recall the second which will be needed later on, and describe a particular case for which composition is commutative.

Lemma 2 *(Associativity). Let U, V, W be three drags sharing no vertices nor variables. Then, there exist two switchboards ζ and ξ for respectively (V, W) and $(U, V \otimes_\zeta W)$ iff there exist two switchboards θ and γ for respectively (U, V) and $(U \otimes_\theta V, W)$ such that $(U \otimes_\theta V) \otimes_\gamma W = U \otimes_\xi (V \otimes_\zeta W)$.*

Furthermore, γ is a rewriting switchboard if ξ, ζ are rewriting switchboards and ξ is a rewriting switchboard if γ, θ are rewriting switchboards.

Lemma 2 is proved in a particular case in [9]. We give here the proof for the general case that is needed later in the proof of Lemma 6. We will need restrictions of $\xi, \zeta, \gamma, \theta$ to some subsets of their domain and target, such as $\xi_{V\to U}$ whose domain is the subset of sprouts of V which are sprouts of $V \otimes_\zeta W$ and image is the list of roots whose corresponding vertices belong to U. Likewise, $\xi_{U\to W}$ is the restriction of ξ_U whose image is the list of roots whose corresponding vertices belong to W. Note that ξ_U is $\xi_{U\to V\otimes W}$ and $\zeta_{V\to W}$ is ζ_V.

Proof. We carry out one direction of these statements, the other having obviously the same proof. We define θ and γ so that they define the same sets of switchboard components as ξ and ζ, hence ensuring that both compositions are identical as we shall show. A difficulty is to show that these definitions are well-behaved injective maps, as required for switchboards. The property of ξ and ζ that makes it all true follows from the fact that the expression $V \otimes_\zeta W$ is computed first in $U \otimes_\xi (V \otimes_\zeta W)$, since occurring inside a pair of parentheses:

$$(\mathcal{D}om(\xi) \cup \mathcal{I}m(\xi)) \cap \mathcal{D}om(\zeta) = \varnothing \qquad (*)$$

The definition of θ, γ, which can be easily followed on Fig. 3, is by cases on the domains of the switchboards ξ, ζ:

- let $s \in \mathcal{D}om(\xi_{U\to V})$. Then, $\theta_U(s) = \xi(s)$;
- let $s \in \mathcal{D}om(\xi_{U\to W})$. Then, $\gamma_{U\otimes_\theta V}(s) = \xi(s)$;
- let $s \in \mathcal{D}om(\xi_{V\to U})$. Then, $\theta_V(s) = \xi(s)$;
- let $s \in \mathcal{D}om(\xi_{W\to U})$. Then, $\gamma_W(s) = \xi(s)$;
- let $s \in \mathcal{D}om(\zeta_{V\to W})$. Then, $\gamma_{U\otimes_\theta V}(s) = \zeta(s)$;
- let $s \in \mathcal{D}om(\zeta_{W\to V})$. Then, $\gamma_W(s) = \zeta(s)$.

It is then easy to verify that θ, γ satisfy (*): $(\mathcal{D}om(\theta) \cup \mathcal{I}m(\theta)) \cap \mathcal{D}om(\gamma) = \varnothing$.

We first show that both compositions define the same drag, that is, that $\theta^*\gamma^* = \zeta^*\xi^*$. Using the switchboards relationally, (*) implies that $(\gamma \cup \theta)^* = \theta^*\gamma^*$ and $(\xi \cup \zeta)^* = \zeta^*\xi^*$. But considered as sets of switchboard components, $\gamma \cup \theta = \xi \cup \zeta$, and we are done.

We show now that θ and γ are switchboards. By their definition by (disjoint) cases, they are maps. θ is injective since so is ξ. For γ, injectivity results from injectivity of ξ and ζ and the assumption that U, V, W do not share vertices. The coherence conditions (1) and (2) follow from the coherence conditions for ξ, ζ and the assumption that the sets of variables of the drags U, V, W are pairwise disjoint. We are left with well-behavedness.

Assume there exists a cycle among the sprouts of U, V, W for either γ or θ. Then, there would exist a cycle among those sprouts involving ξ and ζ. Since and ζ are well-behaved, this cycle must alternate ξ and ζ sequences. By property (*), the only possible sequences of ξ and ζ are of the form, using ξ, ζ relationally, $s\,\zeta^*\xi^*\,t$. But again, (*) imposes that $s \neq t$. So, no cycle using ξ and ζ is possible, and therefore θ and γ must be well-behaved.

We are left showing that γ is a rewriting switchboard if so are ξ, ζ. By its definition, $\gamma_W = \zeta_{W\to V} \cup \xi_{W\to U}$, and since $\xi_{W\otimes_\zeta V}$ is total, $\mathcal{D}om(\xi_{W\to U}) = \mathcal{S}(W) \setminus \mathcal{D}om(\zeta_W)$. It follows that γ_W is total. Now, $\gamma_{U\otimes_\theta V} = \xi_{U\to W} \cup \zeta_{V\to W}$, and since ξ_U and ζ_V must be linear, so is $\gamma_{U\otimes_\theta V}$. And since surjectivity of $\zeta_{V\to W}$ implies surjectivity of $\gamma_{U\otimes_\theta V}$, we are done. □

Remark 1. Note that we do not claim that θ is a rewriting switchboard when so are ξ, ζ. We will not need it, fortunately, since it is not true: $\xi_{U\to V}$ is surjective on the roots of V which are not already eaten by $\zeta_{W\to V}$, but not on all roots of V.

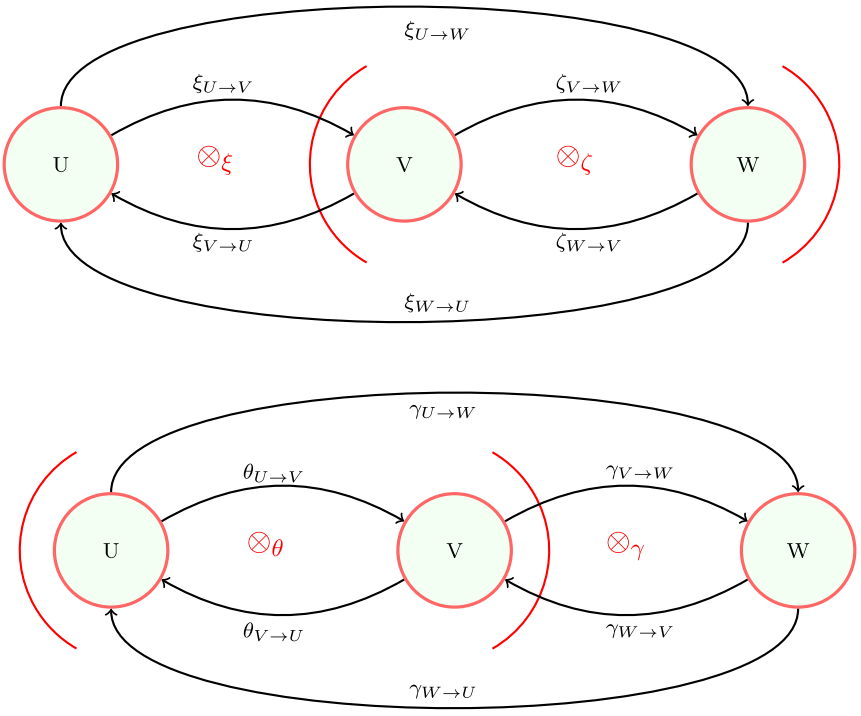


Fig. 3. Proof of associativity.

The composition of two drags D, D' is obviously commutative (modulo a circular permutation of their respective lists of roots):

Lemma 3 *(Commutativity). Let D, D' be two drags sharing no vertices and ξ a switchboard for (D, D'). Then $D \otimes_\xi D' =_\sigma D' \otimes_\xi D$, where σ is a circular permutation of roots which is the identity if ξ is a rewriting switchboard.*

2.4. Drag rewriting

Rewriting with drags is similar to rewriting with trees: we first select an instance of (some renaming 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. First, we define what kind of drags is allowed in rules:

Definition 11. A *pattern* is a clean drag containing no isolated sprout and all of whose vertices have at most one root, i.e. for each vertex u, $R(u) \leq 1$. A *renaming* of a pattern L *away from* a drag D is a renaming L' of the pattern L such that $\mathcal{V}ar(D) \cap \mathcal{V}ar(L) = \varnothing$.

Definition 12 *(Rule).* A *drag rewrite rule* is a pair of clean drags written $L \to R$, such that (i) L is a pattern, (ii) $|\mathcal{R}(R)| = |\mathcal{R}(L)|$, and (iii) $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$.
A set of drag rewrite rules is called a *drag rewriting system*.

A *renaming* of a rule $L \to R$ is a rule $L' \to R'$ such that $L' =^{\iota}_{\alpha} L$ and $R' =^{\iota'}_{\alpha} R$. The renaming $L' \to R'$ is *away from* a given drag D if L' is away from D.

Condition (i) does not show up in [9]. Although it seems restrictive, it is not. Following Fig. 4, assume we need to match a drag $D = h(a)$ which has two roots at vertex h named 1 and 2, with the left-hand side of a rule $L = h(x) \to x$ which has a single root at h named 1. Matching would be straightforward if h (in $h(x)$) had two roots, but is nevertheless possible with a single root: take the extension $\langle z \oplus a, \{z \mapsto 1, x \mapsto 3\}\rangle$, where z has two roots, 1 and 2, and a a single root numbered 3. Then, root transfer will ensure that the result is indeed D (up to drag equality). By exploiting the root transfer mechanism, condition (1) will slightly simplify unification of patterns as well as the confluence section.

Condition (ii) and (iii) ensure that L and R fit with any extension $\langle C, \xi\rangle$ of L, since switchboards map sprouts to positions in a list of roots. Both lists being of the same length ensure that any position in the list R is a position in the list R'. There is however a difficulty to be faced later: the switchboard ξ does not necessarily satisfy well-behavedness with respect to R.

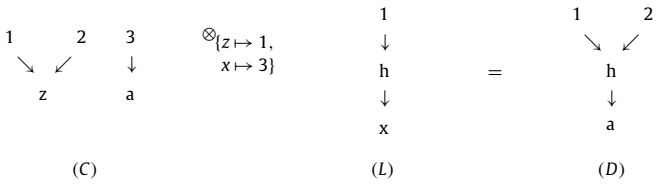


Fig. 4. Patterns and rule matching.

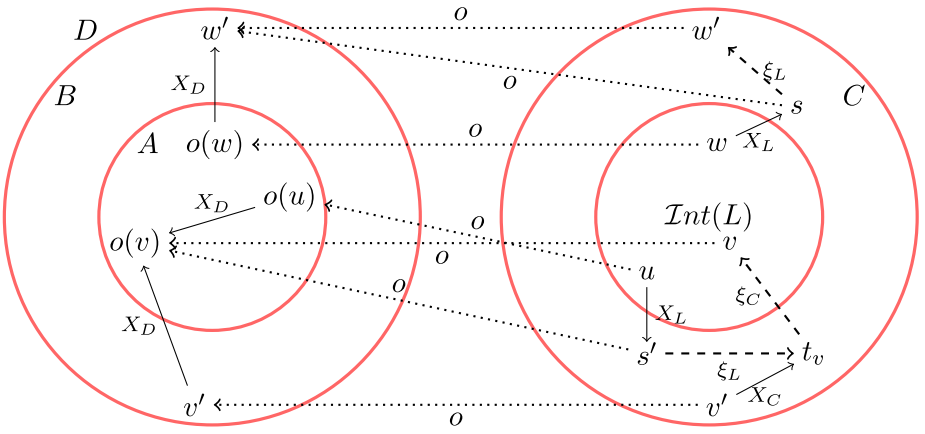


Fig. 5. Proof of Lemma 4.

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

Definition 13. A *rewriting extension* $\langle C, \xi \rangle$ of a clean drag L is *clean* if $C \otimes_\xi L$ is a clean drag.

We now need an important observation absent from [9]:

Lemma 4. *Given a clean drag D and a pattern L, there exists an injection $o : L \to D$ iff there exist a renaming L' of L and a clean rewriting extension $\langle C, \xi \rangle$ of L', called* match *of L' in D at o, such that $D = C \otimes_{\xi'} L'$.*

The composition $C \otimes_\xi L'$ yields a drag whose internal vertices are those of C and L', which explains the need for renaming L, since D and L are both given. Note that the injection o plays the same rôle as a position in the case of trees. We will use this facility when defining rewriting.

Usage of term rewriting systems has sanctified the "match" from a term L' to a term D as being the substitution ξ identifying $L'\xi$ with a subterm of D, the context C obtained by removing $L\sigma$ from D being ignored. Here, we insist that the match is made of both the context C and the switchboard ξ.

Proof. Since D and L have no variable, in common, we can assume w.l.o.g. that all vertices of D are internal.

Given a rewriting extension $\langle C, \xi \rangle$, such that $C \otimes_\xi L = D$, let u be a vertex of L such that $\xi^*(u) = v$. We then define $o(u) = v$. The obtained map o is the identity, hence injective, on internal vertices of L, preserves the successor function, and forces sharing since two sprouts labeled by the same variable x are mapped by ξ^* to the same vertex by the compatibility property of a switchboard. We are left showing that o satisfies property (ii) of injections, that is, an internal vertex v of L is rooted if there exists a new edge in D' whose target is $o(v)$. Since a new edge can only be the result of the composition, this can happen in two different ways, v being necessarily rooted in both cases: some sprout t successor in C of some u is mapped to v by ξ^* resulting in the new edge $(u, o(v))$; or some sprout s successor in L of some u is mapped successively to a sprout t of C by ξ_L and then to v by ξ^*, resulting in the new edge $(o(u), o(v))$.

Conversely, we construct the rewriting extension $\langle C, \xi \rangle$ of L from the given map o as shown at Fig. 5. Let $A = o(\mathcal{I}nt(L))$ be the image by o of the internal vertices of L, and $B = \mathcal{V}er(D) \setminus A$ be its complement, the set of vertices of D which are not the image by o of an internal vertex of L. Vertices in B will be the internal vertices of C so that o can be extended to the internal vertices of C by the identity. Vertices in A are the renamings by o of the internal vertices of L. Edges in D between vertices of B are edges from C; and edges in D between vertices of L are edges from L which may involve a sprout of L mapped to an internal vertex of L by o, a first difficulty. Another difficulty arises with edges in D between a

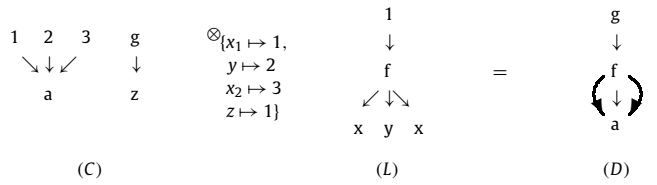


Fig. 6. Injections and rewriting extensions.

vertex of A and one of B, going one way or the other. In all these three cases, the corresponding edges in D will have to be reconstructed by the composition $C \otimes_\xi L$. This requires to appropriately define the sprouts of C and the switchboard ξ.

Let W be a set of fresh sprouts t_v, with $v \in \mathcal{I}nt(L)$ hence $o(v) \in A$, such that one of the following two conditions hold:

(C1) There is a vertex $v' \in B$, hence in C, such that $o(v) \in X_D(v')$
(C2) There is a sprout s in L, such that $o(s) = o(v)$.

We define the set of vertices of C to be $B \cup W$. Before defining the successor function and roots of the drag C, let us define the switchboard ξ as follows (we abuse our notations for simplicity):

- For each sprout t_v in W: $\xi_C(t_v) = v$;
- For each sprout s in L such that $o(s) = o(v) \in A$, $\xi_L(s) = t_v$;
- For each sprout s in L such that $o(s) = w' \in B$, $\xi_L(s) = w'$.

We now define the successor function of C: $\forall v \in B$ such that $X_D(v) = \langle v_1, \ldots, v_k \rangle$, then $X_C(v) = \langle v'_1, \ldots, v'_k \rangle$, where $v'_i = v_i$ if $v_i \in B$, otherwise $v'_i = t_v$.

Finally, we define the roots of C as follows.

- $w' \in B$ has n_s roots in C, where n_s is the number of sprouts of L mapped by o to w';
- $t_v \in W$ has n_s roots in C, where n_s is the number of sprouts in L mapped by o to v.

We can now show that ξ is a switchboard, implying easily that $\langle C, \xi \rangle$ is a clean rewriting extension, the difficult part being that sprouts must be mapped injectively to rooted vertices. Since the rôle of composition is to build new edges, there are three different situations. Blending two of them, we get two cases:

1. there exists a new edge in D of the form $v' X_D\, o(v)$ with $v' \in B$ or of the form $o(w)\, X_D\, o(v)$ (both may happen with the same v). Both require existence of at least two sprouts, $s, \ldots$ of L and t_v of C such that $\xi_L(s) = t_v$ and $\xi_C(t_v) = v$, of predecessors $w, \ldots$ of s in L, and of predecessors $v', \ldots$ of t_v in C. Mapping sprouts to roots injectively is true for ξ_L by definition of the number of roots defined for the vertices in C; for ξ_C, we claim that v has at least one root for mapping t_v to that root, which is a consequence of the new edge property of o.
2. there exists an edge $o(w) X_D\, w'$ in D with $o(w)$ in A and w' in B and sprouts s mapped to w' (both *must* occur for a given w). Indeed, w' is the i-th successor of $o(w)$ in D iff the i-th successor of w in L is a sprout s such that $o(s) = w'$. We are left showing that w' has enough roots for mapping to w' all sprouts s such that $o(s) = w'$, which follows from the definition of the number of roots for w' in C.

Compatibility follows from the property that morphisms force sharing; well-behaved-ness is trivial, as is totality of ξ_L and surjectivity of ξ_C. The verification that $C \otimes_\xi L = D$ can be read on Fig. 5. □

Example 5 *(Example 2 continued).* This example depicted at Fig. 6 illustrates the correspondence between matching and injections described in Lemma 4. Let C be the drag $z \oplus h(z_4)$, z having 3 roots named $1, 2, 3$. Let $\xi = \{x_1 \mapsto 1, y \mapsto 2, x_2 \mapsto 3, z \mapsto 1\}$, where x_1, x_2 denote the two occurrences of x in L. Then, $D = C \otimes_\xi L$. The injection embedding L into D maps the vertex f of L to the vertex f of D, and all three sprouts of L to the vertex a of D as defined in Example 2.

Conversely, let o be that injection. Using the notations of the proof of Lemma 4, we get $A = \{f\}$ and $B = \{g, a\}$. Vertices in $\{x_1, y, x_2\}$ are all mapped to a, and f is mapped to f by o. We get $W = \{t_f\}$ (note that only the first condition is satisfied for generating t_f), $X_C(g) = t_f$, $\xi_C(t_f) = f$, and $\xi_L(x_1) = \xi_L(y) = \xi_L(x_2) = a$. Verification that the composition yields D is left to the reader.

Definition 14 *(Rewriting).* Given a drag rewrite system $\mathcal{R}$, a drag D rewrites at position o to a drag D' with a renaming $L' \to R'$ of the rule $L \to R \in \mathcal{R}$, written $D \longrightarrow^o_{L' \to R'} D'$ or $D \longrightarrow^o_{\mathcal{R}} D'$, if there exist a match $\langle C, \xi \rangle$ of both L' in D at o which is also a match of and R' in D'.

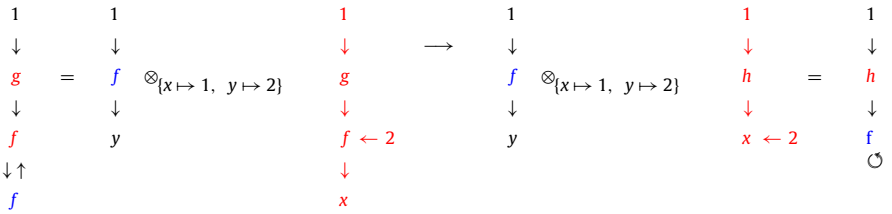


Fig. 7. Rewriting and cycles.

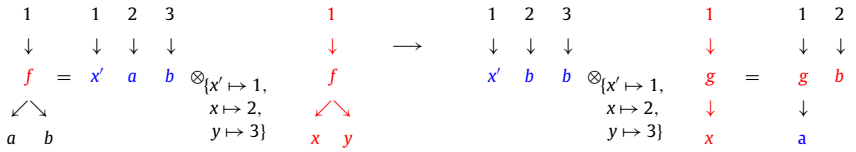


Fig. 8. Rewriting and connected components.

Notice that, according to the previous definition, if D rewrites to D', we have that $\langle C, \xi \rangle$ is a rewriting extension of both, D and D', which means that D' is the result of replacing L' by R' in D.

Remark 2. The assumption that $\langle C, \xi \rangle$ is a rewriting extension for R' does not always follow from the assumption that it is one for L' as one could expect. The point is that the switchboard ξ of the extension does not satisfy all properties needed to be a switchboard for R'. Take for example $f(x) \to x$, with one root on each side. Take now for D a loop on a rooted vertex labeled by f, for C the context reduced to the sprout y with two roots, and $\xi = \{x \mapsto 1, y \mapsto 1\}$. $\langle C, \xi \rangle$ is well-behaved for $f(x)$ but ill-behaved for x since x is mapped by ξ to y and y to x. It turns out that there exists no switchboard ξ well-behaved for both L' and R' that permits rewriting this cycle with this rule, a degenerated case that went unnoticed in [9]. This is why we need to assume in the definition that $\langle C, \xi \rangle$ is a switchboard for both L' and R'. Checking whether the switchboard ξ for L' is also a switchboard for R' is only needed for rules whose right-hand sides is a variable, hence is not really painful in practice.

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.

In the sequel, we adopt the convention that L and R are renamed appropriately, whatever D is, that is, that rules in $\mathcal{R}$ are defined up to renaming of their vertices. We also use $\longrightarrow^*$, as is usual, for the transitive closure of the rewriting relation.

Example 6. In Fig. 7, 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, colors which are reflected in the input term (showing the rule applies across its cycle) and output term.

Example 7. This time, the rooted term $f(a, b)$ in Fig. 8 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.

Lemma 4 is important, since it implies that the result of rewriting a drag at some position o is unique, as it is for trees.

Rewriting drags is of course monotonic with respect to composition, which subsumes monotonicity and stability of rewriting terms:

Lemma 5 *(Monotonicity). Assume that* $D \longrightarrow_{L \to R} D'$ *and let* $\langle C, \xi \rangle$ *be an extension of* D *such that* $C \otimes_\xi D$ *is clean. Then* $C \otimes_\xi D \longrightarrow_{L \to R} C \otimes_\xi D'$.

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

3. Unification

Unification of two terms s, t is somewhat simple: a substitution applied to both identifies them (makes them identical). Assuming s, t share no variable, this substitution is simply the union of two substitutions, one for s and one for t. A substitution is just a particular case of composition as we have seen at Example 7, using a switchboard whose one component is empty, hence our definition of unification will be based on composition: two patterns U, V are unified by composing them with some rewriting extensions $\langle C, \xi\rangle$ and $\langle D, \zeta\rangle$, resulting in the same drag W, same referring here to drag equality modulo renaming.

We could be satisfied with that definition, but we also want to take care of our particular use of unification to characterize drags, called *overlaps*, that can be rewritten in two different ways with two rules $L \to R$ and $G \to D$. In the case of terms, one of L, G stands above the other in the overlap, that is, G is unified with the subterm of L at some position p, or vice versa. If σ is a unifier, the overlap is then either $L\sigma$ or $G\sigma$ (or both if p is the root position). The situation is different with graphical structures, none is above the other, they just share some common subdrag. Two drags U, V are therefore unified at partner vertices $(\overline{u}, \overline{v})$, the solution being a pair of extensions $\langle C, \xi\rangle$ of U and $\langle D, \zeta\rangle$ of V that identifies $C \otimes_\xi U$ and $D \otimes_\zeta V$ at these partner vertices.

Definition 15. Given two drags U, V sharing no vertices, 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).

The two lists of partner vertices $\overline{u}$ and $\overline{v}$ can also be organized as a set of unordered pairs $\{(u_i, v_i)\}_i$. The order between the elements of a pair is not important since one must be in U and the other in V, and U, V share no vertex, hence eliminating any potential ambiguity.

Definition 16. A drag *unification problem*, $U[\overline{u}] = V[\overline{v}]$, is a pair (U, V) of patterns that have been renamed apart, along with partner vertices $P = (\overline{u}, \overline{v})$. A *solution* (or *unifier*) of the drag unification problem $U[\overline{u}] = V[\overline{v}]$ is a pair of rewriting extensions $\langle C, \xi\rangle$ and $\langle D, \zeta\rangle$ of U and V respectively, such that $C \otimes_\xi U$ and $D \otimes_\zeta V$ are *identified below* $(\overline{u}, \overline{v})$, that is:

(i) $C \otimes_\xi U$ and $D \otimes_\zeta V$ are equal modulo $\iota : \mathcal{V}er(C) \to \mathcal{V}er(D)$, i.e., $C \otimes_\xi U =^\iota D \otimes_\zeta V$;
(ii) $\overline{v} = \iota(\overline{u})$;
(iii) $\forall w \in \mathcal{V}er(U) \setminus X^*_U(\overline{u}) : \iota(w) \in \mathcal{V}er(D)$;
(iv) $\forall w \in \mathcal{V}er(V) \setminus X^*(\overline{v}) : \iota^{-1}(w) \in \mathcal{V}er(C)$.

A drag equal modulo renaming to $C \otimes_\xi U$ (hence to $D \otimes_\zeta V$) is called *overlap* of U, V below $(\overline{u}, \overline{v})$.

A unification problem $U[\overline{u}] = V[\overline{v}]$ is *solvable* if it has a solution. We denote by $Sol(U[\overline{u}] = V[\overline{v}])$ the (possibly empty) set of all its solutions.

The overlap drags $W = C \otimes_\xi U$ and $W' = D \otimes_\zeta V$ witness the property that U and V are embedded in W and W' respectively, and that these two embeddings, o and o' coincide at a list of *partner vertices* (condition (ii)) and recursively at their successors (condition (i)), but not at their ancestors which are unreachable from either $\overline{u}$ or $\overline{v}$ (conditions (iii) and (iv)). Note that we could have allowed $W = C \otimes_\xi U$ and $W' = D \otimes_\zeta V$ to be equal modulo an arbitrary renaming including a variable renaming: these two definitions are actually equivalent.

Solutions of a unification problem are defined with the context drag coming first in the products $C \otimes_\xi U$ and $D \otimes_\zeta V$, which is of course consistent with our definitions of rewriting and rewriting extensions. We will stick to this convention in the sequel, even if it does not actually matter since composition is commutative.

Example 8 *(Figs. 9 and 10).* Let $U = f(h(x))$ and $V = g(h(a))$ in Fig. 9, in which U has two roots, f and h in this order, and V has two roots g and h in this order (root numbers of U, V being written in bold face on the figure). Let the partner vertices be $\{(h, h)\}$.

Consider the rewriting extension $\langle C, \xi\rangle$ such that $C = z_1 \oplus g(z_2) \oplus a$ with three roots at z_1, g and a in this order and $\xi = \{z_1 \mapsto 1, z_2 \mapsto 2, x \mapsto 3\}$. Then $C \otimes_\xi U$ is the drag with two roots at f and g in this order sharing the subdrag $h(a)$.

Consider now the rewriting extension $\langle D, \zeta\rangle$ such that $D = f(y_1) \oplus y_2$ with two roots at f and y_1 in this order and $\zeta = \{y_1 \mapsto 1, y_2 \mapsto 2\}$. Then $D \otimes_\zeta V$ is equal to $C \otimes_\xi U$ (hence coincide on the figure), and the pair of rewriting extensions $\langle C, \xi\rangle$, $\langle D, \zeta\rangle$ is therefore a solution.

Note that flipping the two roots of V would give a different unification problem, whose solution would simply require to slightly change ζ: changing the order of roots in either U or V does not alter unifiability.

Let us now consider Fig. 10, with drags U and V as in the previous case, except that $V = f(h(a))$, with partner vertices $\{(h, h)\}$, as before. In this case, the drag $W = f(h(a))$, with two roots on f, would not be an overlap of U and V below $\{(h, h)\}$, i.e., it will not define a correct solution to this unification problem. The reason is that conditions (iii) and (iv) of Definition 16 would not be satisfied, because of identifying the two vertices f, which are above the partner vertices. That is, the correct solution, depicted in Fig. 10, should not identify these vertices. □

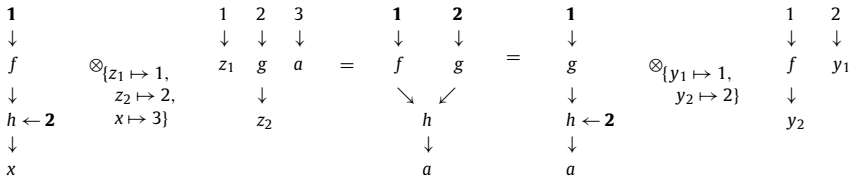


Fig. 9. Overlap drag.

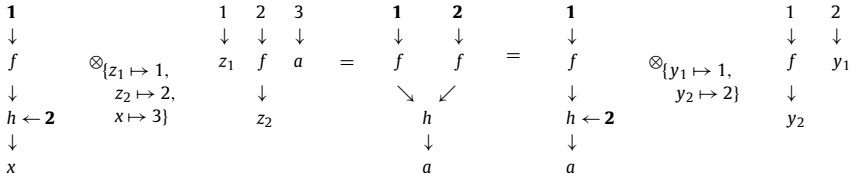


Fig. 10. Another overlap drag.

Indeed, we want unification to be minimal, that is, to capture all possible extensions that identify U and V. 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 of solutions. In a second subsection, we show that unification of drags is unitary, as for terms and dags.

3.1. Subsumption

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

Note that V being clean, U must be clean as well by definition of a rewriting extension.
In the following, we assume for convenience that the sprouts of U, V are labeled by different sets of variables.

Lemma 6. *$\succeq$ is a quasi-order on clean drags, called* subsumption*, whose strict part is a well-founded order. Two clean drags are equivalent modulo subsumption iff they are equal modulo variable renaming and the number of roots of their rooted vertices coincide.*

Proof. The relation $\succeq$ being reflexive, we show transitivity. Let U, V, W be three clean drags whose sprouts are labeled 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 rewriting extensions $\langle C, \xi\rangle$ of V and (D, ζ) of W. By Lemma 2, $U = E \otimes_\theta W$, for some rewriting extension $\langle E, \theta\rangle$ of W, hence $U \succeq W$.

Assume that $U \succeq V \succeq U$, hence $U = C \otimes_\xi V$ and $V = D \otimes_\zeta U$, using the same notations as above. It follows that C and D have no internal vertex, hence are a bunch of sprouts. Therefore, U and V have the same internal vertices, while their sprouts correspond bijectively. Further, U and V must have the same (modulo renaming) rooted vertices, since a rootless vertex cannot become rooted by composition, but the number of roots of a rooted vertex can be increased by composition, or decreased down to zero.

Assume now that $U \succ V$. Then, either $|U| > |V|$, or $|U| = |V|$. In the latter case, they cannot have the same number of variables labeling their sprouts, since otherwise U and V would be identical up to variable names, hence contradicting our assumption. Since $U \succeq V$, then $U = C \otimes_\xi V$ where C cannot have internal vertices since $|U| = |V|$, and ξ cannot be bijective, otherwise $U \simeq V$. Hence ξ maps at least two variables of V to a same (rooted) variable of C, which becomes a variable of U in the composition. Well-foundedness follows, since the number of variables labeling the sprouts 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. The existence of cycles in drags makes it however impossible, in general, to separate the substitution from the context. 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.

Given a clean drag D whose one vertex u is rooted, it is always possible to add new roots at u by composition with a rewriting extension, namely the rewriting extension $\langle z, \{z \mapsto u\}\rangle$, where z is a fresh many-rooted sprout. It is possible as well to remove a root (if u is accessible from some other vertex in D in case it has a single root) by composition with the rewriting extension $\langle z, \{z \mapsto u\}\rangle$ where z is a fresh rootless sprout. On the other hand, a single-rooted vertex u that is not accessible from any other vertex cannot loose its only root by composition with a rewriting extension. So, there may

be many equivalent rewriting extensions of a pattern L, whose compositions with L will only differ in the number of roots of the vertices of the resulting drags. This will be used to ease the construction of most general unifiers, by choosing the number of roots that makes unification easiest.

3.2. Marking algorithm for unification

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.

More precisely, we describe a *marking algorithm* that computes an equivalence relation between the vertices of two drags U, V to be unified, from which their most general unifier is extracted in Section 3.6 if no failure occurs. The algorithm consists of a set of transformation rules operating on the drag $U \oplus V$, where some pairs of vertices may already hold a mark, meaning that they are equivalent and that they should be identified by any solution of the given unification problem. The rules construct this equivalence by marking new pairs of vertices of $U \oplus V$, in the style of Patterson and Wegman unification algorithm [31]. A related idea appears even earlier in [22]. Our treatment is very close to the latter. The algorithm includes also failure rules that return the special expression $\bot$.

Identifying $C \otimes_\xi U$ and $D \otimes_\zeta V$ 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. Since C, D are yet unknown, this propagation takes place on vertices of U and V, hence on the drag $U \oplus V$. 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 (in practice, we can mark it in blue from the beginning). To ensure freshness, we shall memorize the number c of pairs of vertices that have been marked so far, and increment c by one at each use of a mark. The drag $U \oplus V$ in which some pairs of vertices hold a same mark is called a *marked unification problem*. Two vertices u, v of a marked unification problem $U \oplus V$ (sometimes denoted $U \oplus V[u][v]$), are on the *same side* if they both belong to either U or V, and on *opposite sides* otherwise.

Propagation (rule *Propagate*) computes therefore a succession of marked unification problems, denoted by $U \oplus_m V$, starting with the marked unification problem $U \oplus_0 V$ whose marked vertices are exactly the partner vertices. Propagation will stop when there are no more pairs of internal vertices holding a red mark, unless one of the following two situations occurs: (i) two sprouts v, w hold the same variable; (ii) some vertex u marked with both $\mathbf{i}$ and $\mathbf{j}$ provides a link between two other different vertices v and w marked $\mathbf{i}$ and $\mathbf{j}$ respectively.

In both cases, the pair of vertices (v, w) must now be marked by rules *Variable case* and *Merge*, respectively, if not marked already, giving then possibly rise to new propagations rules.

When no rule is applicable, the procedure stops at some step k, where some of the vertices of $U \oplus V$ are marked and the others unmarked. At that point, an internal vertex u in $U \oplus V$ is said to be *singular* if it doesn't share a mark with another internal vertex. Vertices that are unreachable from the partner vertices are particular singular vertices, but some reachable internal vertices may also be singular. Note that singular vertices may share marks with sprouts.

Failure rules detect situations where unification of $U \oplus_k V$ is not possible. There are three of them: (i) two internal vertices sharing a red mark hold a different label (rule *Symbol conflict*); (ii) two internal vertices of the same drag share a red mark, since a unifying solution cannot identify them (rule *Internal conflict*); (iii) the absence of root at a given vertex makes it impossible to build a unifying solution from the resulting marked unification problem (rule *Lack of root*). This is the case when a sprout s and an internal vertex u of the same drag share the same red mark. We then need to identify them, implying that all edges incoming to s should be transferred to u, hence requiring that u is rooted. The other case is when no root is available to mimic singular vertices on the side where they are missing.

We assume that vertices singled out in the precondition of a rule are pairwise different, and that a pair of vertices sharing a mark is never marked again.

Definition 18 *(Marking algorithm).* Given a unification problem $U[u_1, \ldots, u_n] = V[v_1, \ldots, v_n]$, the marking algorithm computes a sequence of marked unification problems $U \oplus_0 V, \ldots U \oplus_m V$, where $U \oplus_0 V$ is the result of marking u_1, v_1 with $\mathbf{1}$, $\ldots$, and u_n, v_n with $\mathbf{n}$. Then, $U \oplus_{m+1} V$ $(m \geq 0)$ is defined by the application to $U \oplus_m V$ of one of the 7 rules given below.

We assume that c is the number of pairs of vertices that have been marked so far, and that $u : h \cdot \mathbf{i_1} \cdot \cdots \cdot \mathbf{i_j}$ denotes the vertex u labeled with h in the drag $U \oplus V$ and holding the marks $\mathbf{i_1}, \ldots, \mathbf{i_j}$, painted in either blue or red, in the drag $U \oplus_m V$.

1. *Propagate*: If $u : f \cdot \mathbf{i}, v : f \cdot \mathbf{i} \in U \oplus_m V$ where f has arity k, $s_1, \ldots, s_k$ are the k successors of u and $t_1, \ldots, t_k$ are those of v, then $U \oplus_{m+1} V$ is obtained from $U \oplus_m V$ by marking the pairs (s_1, t_1) with $\mathbf{c+1}$, $\ldots$, (s_n, t_n) with $\mathbf{c+n}$, and turning the mark $\mathbf{i}$ to blue.
2. *Variable case* If $s : x \cdot \mathbf{i}, u : f \cdot \mathbf{i} \in U \oplus_m V$, then $U \oplus_{m+1} V$ is obtained from $U \oplus_m V$ by turning the mark $\mathbf{i}$ to blue.
3. *Merge*: If $s : x, t : x \in U \oplus_m V$, then $U \oplus_{m+1} V$ is obtained by marking s, t with $\mathbf{c+1}$.
4. *Transitivity*: If $u : f \cdot \mathbf{i} \cdot \mathbf{j}, v : f \cdot \mathbf{i}, w : f \cdot \mathbf{j} \in U \oplus_m V$, then $U \oplus_{m+1} V$ is obtained by marking v, w with $\mathbf{c+1}$.
5. *Symbol conflict*: If $u : f \cdot \mathbf{i}, v : g \cdot \mathbf{i} \in U \oplus_m V$ and $f \neq g$, then $U \oplus_{m+1} V$ is $\bot$.
6. *Internal conflict*: If $u : f \cdot \mathbf{i}, v : f \cdot \mathbf{i} \in U \oplus_m V$, and u and v are on the same side, then $U \oplus_{m+1} V$ is $\bot$.

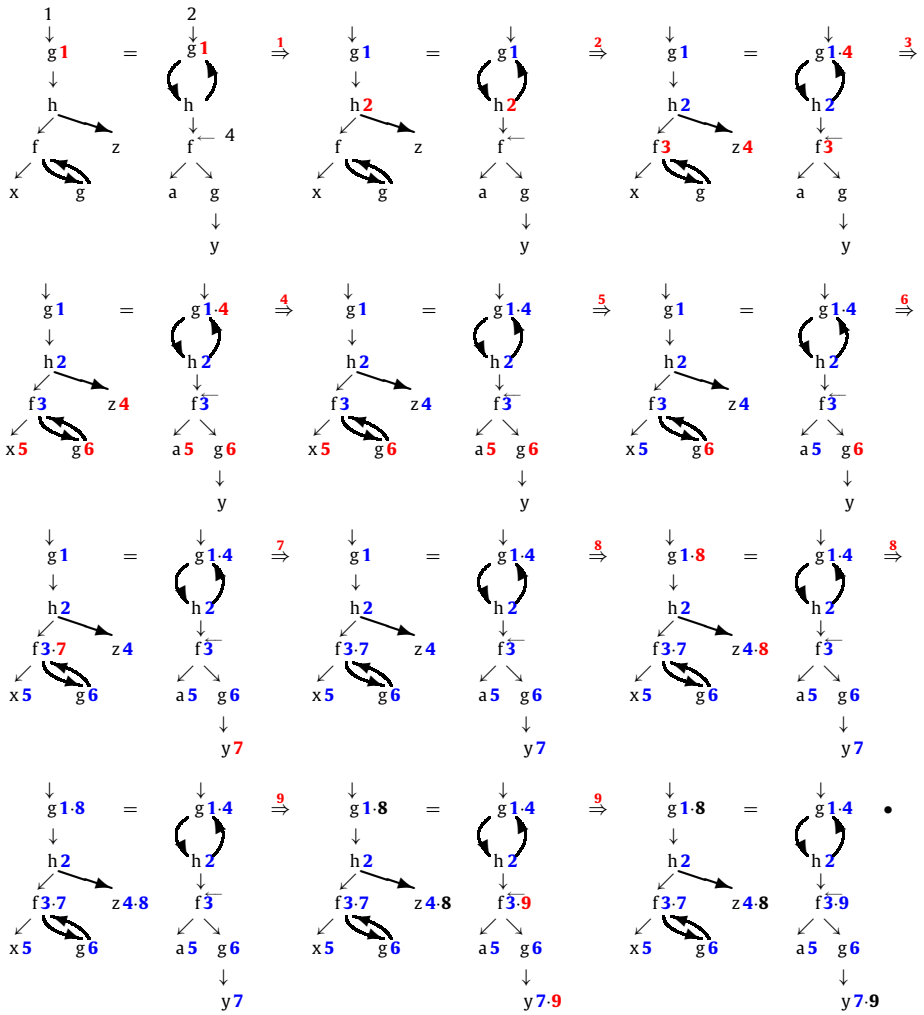


Fig. 11. Successful unification of two patterns.

7. *Lack of root*: If $u : f \cdot \mathbf{i}$, $v : f \cdot \mathbf{i} \in U \oplus_m V$, u is a rootless internal vertex and
   – v is a sprout on the same side as u, or
   – u and v are on opposite sides and there is a singular vertex w in $U \oplus_m V$ whose one successor of v,
   then $U \oplus_{m+1} V$ is $\bot$.

Remark 3. Some rules are reminiscent from the unification rules for terms [7], although we don't use the same rule names except for *Merge*. For example, we use *Propagate* here rather than *Decompose* to stress the fact that drags cannot be treated as terms. The failure rules also depend on the roots present in an equivalence class, since drag equality checks their number at all pairs of corresponding vertices.

Remark 4. Unification of finite terms differs from unification of infinite rational terms by only one rule called occur-check. Since terms and rational terms are two particular cases of drags, one might expect that the occur-check rule applies in case the occur-check cannot be solved by forming a cycle. This is indeed a particular case of the first alternative in *Lack of root*.

Example 9. In our example of Fig. 11, unification of the initial two drags proceeds in eleven steps and succeeds. *Propagation* steps are labeled by the red mark processed. For instance, in the first step, we apply the *Propagation* rule to the pair of vertices $g \cdot \mathbf{1}$. As a consequence, their successors labeled h are marked **2** and the marks in the two vertices $f \cdot \mathbf{1}$ are now blue. The same happens with the second step, where *Propagation* is applied to the pair of vertices $h \cdot \mathbf{2}$, causing that their successors, the two vertices labeled f on the one hand, and vertices labeled z, g on the other hand, are marked **3** and **4**, respectively.

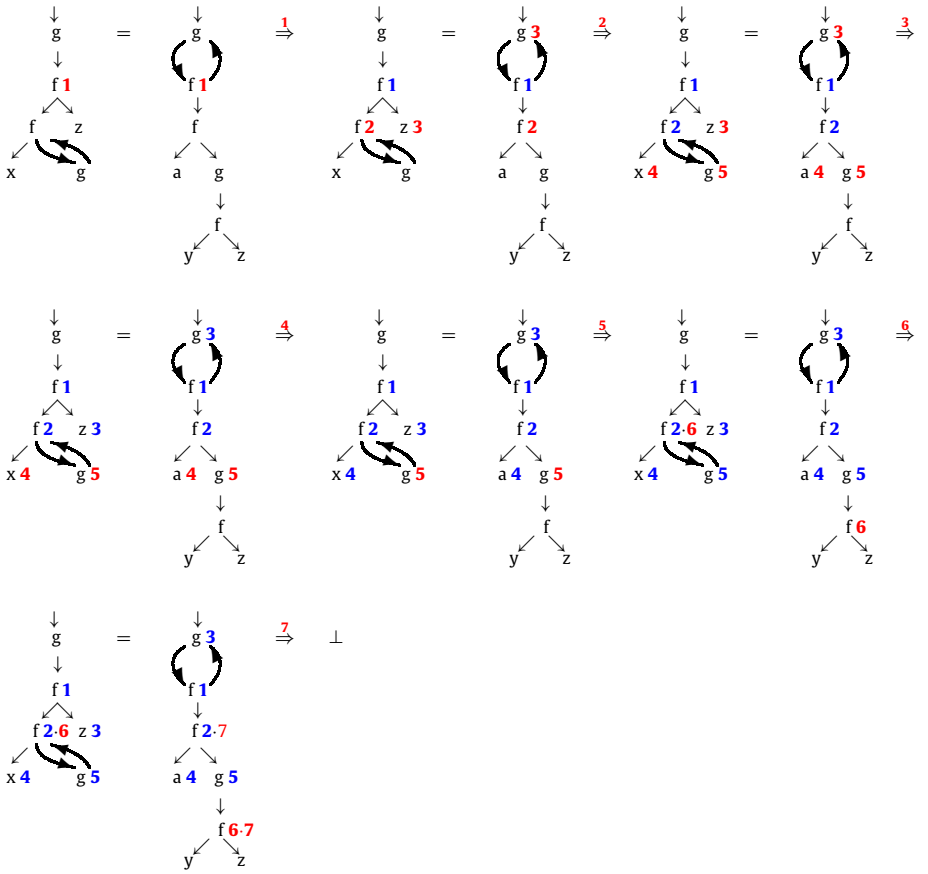


Fig. 12. Unification of two patterns (failure case).

In the case of *Transitivity*, steps are labeled by the generated mark. This explains why some steps have the same label. For instance, this happens with the two steps labeled **9**, where the first one is the application *Transitivity* step to vertices $f \cdot \mathbf{3} \cdot \mathbf{7}$, $f \cdot \mathbf{3}$, and $y \cdot \mathbf{7}$. As a consequence, mark **9** is added to $f \cdot \mathbf{3}$ and $y \cdot \mathbf{7}$.

When a red mark labels a sprout, we apply the *Variable case* rule, as in step **4**, and the mark is simply turned blue.

Example 10. An example of failure is given at Fig. 12. The first 5 steps are all *Propagation* or *Variable case* steps. Step 6 is a *Transitivity* step and step 7 an *Internal conflict, since the two vertices marked* **7** are on the same side and both are internal.

Note that we violate our definition of a unification problem by having a common variable z across the "=" sign. We could of course have two different variables z, z', and a third successor to the f vertex, a sprout labeled z on the left, and a sprout labeled z' on the right. We would then satisfy the constraint to the price of a few more steps before finding the failure. Carrying out the precise calculation in this case is left as an exercise.

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

Lemma 7. *Unification rules terminate.*

Proof. Since a pair of identical vertices is never marked, a pair of different vertices is never marked twice, and added sprouts take place of unreachable vertices that are never marked, the number of marked vertices of a unification problem $U[u] = V[v]$ is at most equal to $(|U| + |V|) \times (|U| + |V| - 1)$. □

3.3. Unification congruences

Correctness of a set of unification rules is the property that the solutions of a unification problem are preserved by application of the rules, until some normal form is obtained which contains them all. Defining precisely what preservation means is the problem we tackle now.

As a general fact, congruences are at the heart of unification and of unification algorithms. In our case, solutions define congruences, and markings define congruences as well. Preservation then relates both kinds of congruences, those defined by markings being coarser than those defined by solutions.

The notion of congruence on terms applies to drags directly:

Definition 19. An equivalence relation $\equiv$ on the set of vertices of a drag $U \oplus V$ is a *congruence* if it satisfies the following properties:

1. any two equivalent internal vertices u, v have identical labels;
2. the successors of equivalent internal vertices are pairwise equivalent;
3. sprouts with identical label are equivalent.

The main difference between terms and drags is that the latter may have cycles, hence a sprout can be equivalent to any other vertex in a given drag while it cannot in a term.

We now define the congruence associated with the solutions of a given unification problem:

Definition 20. Let $U[\overline{u}] = V[\overline{v}]$ be a unification problem. To any solution $S = (\langle C, \xi\rangle, \langle D, \zeta\rangle)$ such that $C \otimes_\xi U[\overline{u}] =^\iota D \otimes_\zeta V[\overline{v}]$, we associate the least equivalence $=_S$ on the vertices of $U \oplus V$ such that:

1. $w \in \mathcal{V}er(U)$, $w' \in \mathcal{V}er(V)$: then $w =_S w'$ if $\zeta^*(w') = \iota(\xi^*(w))$;
2. $w \in \mathcal{V}er(U)$, $w' \in \mathcal{S}(U)$: then $w =_S w'$ if $\xi^*(w') = \xi^*(w)$;
3. $w \in \mathcal{V}er(V)$, $w' \in \mathcal{S}(V)$: then $w =_S w'$ if $\zeta^*(w') = \zeta^*(w)$.

Unification equivalence $=_{unif}$ is now defined as the intersection $\bigcap_{S \in Sol(U[\overline{u}]=V[\overline{v}])} =_S$.

Note that, if w, w' are both sprouts of U (or of V) holding the same variable, then they must be sent to the same vertex by ξ (or by ζ), hence they are equivalent.

Lemma 8 *(Unification congruence). Given a solution S of a unification problem $U[\overline{u}] = V[\overline{v}]$, $=_S$ is a congruence on the vertices of $U \uplus V$ such that $\forall i \in [1..|\overline{u}|] : u_i =_S v_i$, and $=_{unif}$ is a congruence generated by the set of partner vertices.*

Proof. First, $=_S$ is an equivalence associated with a solution, it is therefore a congruence.

Further, $\xi^*(u_i) = u_i$ and $\zeta^*(v_i) = v_i$ since u, v are internal, and $v_i = \iota(u_i)$ since S is a solution. It follows then that $\equiv_{unif}$ is the least congruence satisfying this same property, hence is generated by the partner vertices. □

Since unreachable ancestors of partner vertices cannot be identified by a solution, it follows that the unification congruence of a solvable unification problem does not contain unreachable ancestors of its partner vertices.

We now define the congruence computed by the unification rules:

Definition 21. Given a marked unification problem $U \oplus_k V$, we denote by $\equiv_k$ the binary relation 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 *marking equivalence* $\equiv_{unif}$ as $\bigcup_k \equiv_k$.

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

Lemma 9. *$\equiv_k \subseteq \equiv_l$ for all $l \geq k$ such that $\equiv_l$ is defined.*

It follows that $\equiv_{unif}$ coincides with $\equiv_n$ defined by $U \oplus_n V$, the obtained normal form of $U \oplus_0 V$ at step n. We believe that this normal form is unique, a property not needed here.

Lemma 10 *(Marking congruence). $\equiv_{unif}$ is a congruence on the vertices of $U \oplus V$ generated by $(\overline{u}, \overline{v})$.*

Proof. By definition, $\equiv_k$ is symmetric, hence is an equivalence thanks to *Transitivity*. Hence $\equiv_{unif}$ is an equivalence. Since unification has terminated with success, *Propagation* and Lemma 9 ensures the first two properties of a congruence, and *Merge* and Lemma 9 the third. Finally, *Initialization* and Lemma 9 ensure the last required property. □

It should by now be clear that, although they are defined quite differently, $=_{unif}$ and $\equiv_{unif}$ are nevertheless the same congruence on the vertices of $U \oplus V$, and that's why we adopted a very similar notation. The proof of this key property is the matter of the next two sections.

3.4. Soundness of the unification rules

Soundness is the property that the marking algorithm succeeds iff the unification problem is solvable. Soundness is weaker than correctness, since it could be that some solutions are lost during the unification process.

We split the soundness lemma into two, depending on whether the unification problem is solvable or unsolvable.

Lemma 11. *Let $U[\overline{u}] = V[\overline{v}]$ be a solvable unification problem, and $S = (\langle C, \xi\rangle, \langle D, \zeta\rangle)$ a solution such that $C \otimes_\xi U[\overline{u}] =^\iota D \otimes_\zeta V[\overline{v}]$ for some map ι. Then, $\forall k : \equiv_k \subseteq =_S$.*

Proof. By induction on k.

- $u \equiv_0 v$. Then, u, v are two partner vertices, hence $u =_S v$ by Lemma 10.
- $u \equiv_{k+1} v$. If $u \equiv_k v$, we conclude by induction on k. Otherwise, $U \oplus_{k+1} V[u \cdot c + 1][v \cdot c + 1]$. If $u = v$, the result holds since $=_S$ is an equivalence. Otherwise, there are three cases (*Variable case* implies $u \equiv_k v$), depending on the rule used to generate the marked pair:
  - *Propagate*: there exist internal vertices u', v' and $i \leq k$ such that $U \oplus_k V[u' : f \cdot i][v' : f \cdot i]$, and u, v are corresponding successors of u', v' in $U \oplus V$. By induction hypothesis, $u' =_S v'$. Hence $u =_S v$ since $=_S$ is a congruence by Lemma 10.
  - *Merge*: u, v are different sprouts such that $U \oplus_k V[u : x][t : x]$. Since, U, V don't share vertices nor variables, u, v must belong both to either U or V. Then $u =_S v$ by definition of a switchboard.
  - *Transitivity*: u, v are different vertices such that $U \oplus_k V[w \cdot i \cdot j][u \cdot i][v \cdot j]$. By induction hypothesis, $w =_S u$ and $w =_S v$. Hence $u =_S v$ since $=_S$ is an equivalence. □

Lemma 12. *Assume that $\equiv_{k+1} = \ \perp$ for some k. Then $U[\overline{u}] = V[\overline{v}]$ is an unsolvable unification problem.*

Proof. Assume by contradiction that the problem is solvable. By Lemma 11, any solution of the unification problem is a solution at step k. We discuss the three possible cases of failure in turn, showing that each one leads to a contradiction, that is, there is no solution at step k:

1. *Symbol conflict*: because $=_S$ is a congruence.
2. *Internal conflict*: because composition cannot identify different internal vertices.
3. *Lack of root*: assume there exists C, D, ξ, ζ such that $C \otimes_\xi U$ and $D \otimes_\xi V$ are identified at step k. We assume w.l.o.g. that u is a vertex of U. There are two cases, one for each failure condition.
   Assume there is a sprout s in U sharing a mark with u, hence $\xi^*(u) = u$ and $\xi^*(s)$ are the same vertex in $C \otimes_\xi U$. Then $\xi^*(s) = u$, requiring that u is rooted, a contradiction.
   Assume there are vertices v, w in V such that v shares a mark with u in U, and w is a singular vertex predecessor of v. Then, $\xi^*(u) = \iota \circ \zeta^*(v)$, hence there would be an edge in $C \otimes_\xi U$ mimicking the edge from $\zeta^*(w) = w$ to $\zeta^*(u)$. Since w is singular in $U \oplus V$, the vertex corresponding to w in $C \otimes_\xi U$ must belong to the context C by definition of a solution and Lemma 8. Therefore, u has one more incoming edge in $C \otimes_\xi U$ than in U, which is impossible since u is rootless. □

3.5. Solved forms

We show first here that the unification rules return an equivalence in *solved form* from which a most general unifying extensions can then be constructed in Section 3.6. This will therefore give us *completeness*, that is the property that the unification algorithm captures all solution of a given unifiable unification problem.

Definition 22 *(Solved form).* Given a solvable unification problem $U[\overline{u}] = V[\overline{v}]$, a congruence $\equiv$ on $U \oplus V$ containing the partner vertices $(\overline{u}, \overline{v})$ is in *solved form* iff

1. no two internal vertices of U or of V are equivalent
2. An internal vertex u must be rooted in the following cases:
   (a) there exists a sprout s equivalent to u in the same drag;
   (b) u is equivalent to a vertex v of the other drag whose one predecessor is singular.

Note that the conditions for a congruence containing the partner vertices to be in solved form are directly inherited from the application conditions of the failure rules.

Non-trivial equivalence classes of a congruence in solved form have a specific structure:

Lemma 13. *Given two patterns U, V, let $\equiv$ be a congruence in solved form on the vertices of $U \oplus V$ that contains the partner vertices $(\overline{u}, \overline{v})$. Then a non-trivial equivalence class is a set of vertices $S = \{u_i\}_{i=1}^{i=m} \cup \{v_j\}_{j=1}^{j=n}$ such that $m, n \geq 1$ and $\forall i < m \forall j < n$, u_i and v_j are sprouts.*

Proof. Follows directly from Definition 22. □

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

We now show that the unification rules deliver solved forms:

Lemma 14. *Assuming the unification problem $U[\overline{u}] = V[\overline{v}]$ does not fail, the equivalence $\equiv_{unif}$ defined on the vertices of $U \oplus V$ by a marked unification problem in normal form is the least congruence in solved form generated by $(\overline{u}, \overline{v})$.*

Proof. By Lemma 7, $\equiv_{unif}$ is well defined, and by Lemma 10, it is a congruence generated by $(\overline{u}, \overline{v})$. It is the least such one generated by these partner vertices, since it contains them and any congruence is closed under *Propagate*, *Transitivity*, and *Merge*.

We are left showing that a failure rule applies to unification problems when $\equiv_{unif}$ is not in solved form, contradicting the assumption of a successful unification.

1. Let $U \oplus V[u][v]$ such that $u \equiv_{unif} v$, $u, v \in \mathcal{I}nt(U)$ and $u \neq v$. By definition of $\equiv_{unif}$, there exists some $k > 0$ such that $U \oplus_k V[u \cdot i][v \cdot i]$ for some i, hence *Internal Conflict* applies at all steps from $k + 1$.
2. Assume the equivalence class of u contains a single other vertex $v \in \mathcal{I}nt(V)$ which is rooted. By definition of $\equiv_{unif}$, there exists $k > 0$ such that $U \oplus_k V[u \cdot i][v \cdot i]$ for some i. By assumption, the class of v contains the same two elements at all steps $k' \geq k$. Hence *Root conflict* applies to the result of unification.
3. Let $u \in \mathcal{I}nt(U)$ be rootless. We proceed by contradiction, showing in both sub-cases that a failure rule applies to the result of unification, contradicting the assumption of success. Assume there exists a vertex v equivalent to u which is either a sprout of U or the successor in V of a vertex w which is singular. By definition of $\equiv_{unif}$, there exists some $k > 0$ such that $U \oplus_k V[u \cdot i][v \cdot i]$ for some i, hence *Lack of root* applies at step k by Lemma 12. Since $U \oplus V$ remains unchanged during the monotone marking process, *Lack of root* applies at all steps $k' \geq k$, hence to the result of unification. □

3.6. Construction of the most general unifying extensions

We now show that a solved form is always solvable, hence their name. Here, the input is a congruence in solved form, which can be seen as a specific unification problem. We therefore construct a most general unifying extension for that solved form.

Definition 23. [mgu] Given a unification problem $U[\overline{u}] = V[\overline{v}]$ and an equivalence $\equiv$ on the vertices of $U \oplus V$ which is in solved form, we define the unifying extensions $\langle C, \xi \rangle$ of U and $\langle D, \zeta \rangle$ of V, as well as the renaming $\iota : C \otimes_\xi U \to D \otimes_\zeta V$ such that $C \otimes_\xi C =^\iota D \otimes_\zeta V$.

Let $S = \{u, s_1, \ldots, s_{m_S}; t_1, \ldots, t_{n_S}, v\}$ be an equivalence class containing internal vertices u from U and v from V that are possibly absent, $m_S \geq 0$ sprouts $\{s_i\}_i$ originating from U and $n_S \geq 0$ sprouts $\{t_j\}_j$ originating from V. The construction is by case on the form of S.

At step 1, we set up an infrastructure of fresh sprouts in C and D that will serve connecting C, D with U, V and ensuring that each vertex in the composition has zero or one root. At step 2, we define the mapping ι. At step 3, we define the successor functions for C and D.

1. For each rooted internal vertex u in U (respectively, $v \in V$) belonging to some class S, we include a fresh sprout $s_S : x_S$ in C, (resp. $t_S : y_S$ in D), with $i + m_S$ roots (resp., $i + n_S$ roots), where i is the number of roots of the internal vertex v from V (resp., u from U) belonging to S if there is one, otherwise 1.
   For each rooted sprout r in U (resp., in V) such that r belongs to the equivalence class of an internal vertex in U (resp. in V), we include a fresh rootless sprout s_r in C (resp., t_r in D).
   Define $\xi_C(s_S) = u$, $\xi_C(s_r) = r$, and for each sprout s_i in S, $\xi_U(s_i) = s_S$
   (resp., $\zeta_D(t_S) = v$, $\zeta_D(t_r) = r$, and for each sprout t_j in S, $\xi_V(t_j) = t_S$).
   % the root of s will disappear in the composition, while the edges of C ending in s_S will then end up in u.
2. - For each class S containing internal vertices u, v from U, V respectively, define $\iota(u) = v$.
   - For each class S containing a (necessary single) singular internal vertex u from U (resp., v from V), include in D (resp., C) a fresh internal vertex u_S, equipped with $n_S + r$ roots, where r is the number of roots, zero or one, of u, with $\mathcal{L}_D(u_S) = \mathcal{L}_U(u)$ (resp., v_S equipped with $m_S + r'$ roots, r' is the number of roots of v, and $\mathcal{L}_C(v_S) = \mathcal{L}_V(v)$); define $\iota(u) = u_S$ (resp., $\iota(v_S) = v$), and for each sprout t_j in S, $\zeta_V(t_j) = u_S$ (resp., for each sprout s_i in S, $\xi_U(s_i) = v_S$);
   - For each class S containing no internal vertex, include two sprouts s_S in C and t_S in D both labeled x_S, equipped with $1 + m_S$ and $1 + n_S$ roots respectively. Define $\iota(s_S) = t_S$, and for each sprout s_i in S, $\xi_U(s_i) = s_S$ (for each sprout t_j in S, $\zeta_V(t_j) = t_S$).

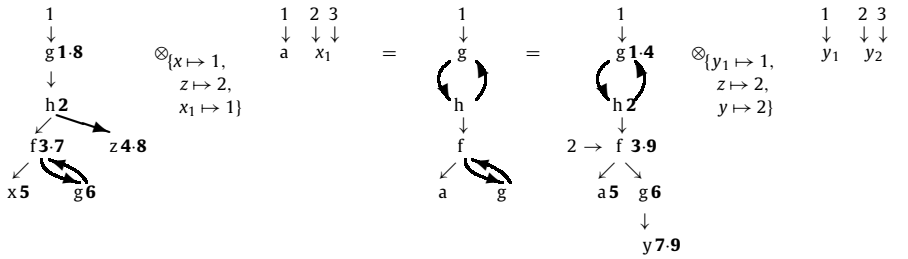


Fig. 13. Most general unifier.

3. For each internal vertex v_S in C (resp., u_S in D) associated with the class S, let $X_V(\iota(v_S)) = \langle v_1 \dots v_k \rangle$ (resp., $X_U(\iota(u_S)) = \langle u_1 \dots u_k \rangle$). We define $X_C(v_S)$ (resp. $X_D(u_S)$) as the tuple $\langle w_1 \dots w_k \rangle$, where, denoting by S_i the class of v_i (resp., u_i):
   - $w_i = \iota^{-1}(v_i)$ if $\iota^{-1}(v_i)$ is in C, (resp., $w_i = \iota(u_i)$ if $\iota(u_i)$ is in D),
   - $w_i = s_{S_i}$ (resp., $w_i = t_{S_i}$), if $\iota^{-1}(v_i)$ is in U (resp. $\iota(u_i)$ is in V).

Before showing that we have defined a solution, we develop two examples. In both cases, the given congruence in solved form is obtained from the marking congruence resulting from applying the unification algorithm. The solution obtained is therefore the most general one for the starting unification problem, not only for the solved form.

Example 11. Fig. 13 shows the two marked drags obtained at Fig. 11 by our unification algorithm, as well as the context drag, switchboard, and overlapping drags obtained by composition with the two marked drags.

The equivalence on vertices is in solved form and has 5 classes, whose elements are given by the name of their drag (U or V, we assume that U is the drag on the left and V on the right), their label and their marks. For instance, $U[g \cdot \mathbf{1}\cdot\mathbf{8}]$ denotes the vertex on top of the first drag on the left, and $V : [\cdot\mathbf{1}\cdot\mathbf{4}]$ denotes the vertex on top of the fourth drag in the figure. The five classes are:

$(U[g \cdot \mathbf{1} \cdot \mathbf{8}], V[g \cdot \mathbf{1} \cdot \mathbf{4}], U[z \cdot \mathbf{4} \cdot \mathbf{8}])$
$(U[g \cdot \mathbf{2}], V[g \cdot \mathbf{2}])$
$(U[f \cdot \mathbf{3} \cdot \mathbf{7}], V[f \cdot \mathbf{3} \cdot \mathbf{9}], V[y \cdot \mathbf{7} \cdot \mathbf{9}])$
$(U[x \cdot \mathbf{5}], V[a \cdot \mathbf{5}])$
$(U[g \cdot \mathbf{6}], V[g \cdot \mathbf{6}])$

According to step 1, we include variables x_1 in C and y_1, y_2 in D, corresponding to the rooted vertices $U[g \cdot \mathbf{1} \cdot \mathbf{8}]$, $V[g \cdot \mathbf{1} \cdot \mathbf{4}]$, and $V[f \cdot \mathbf{3} \cdot \mathbf{9}]$, respectively. Moreover, x_1 has three roots since the class of $U[g \cdot \mathbf{1} \cdot \mathbf{8}]$ includes one rooted vertex in V and one sprout in U. Similarly, y_1 has one root and y_2 has two. Since there are no rooted sprouts in U or V, no additional sprouts are added to C or D. Accordingly, we define $\xi_C(x_1) = 1; \xi_U(z) = 2; \zeta_D(y_1) = 1; \zeta_D(y_2) = 2; \zeta_V(y) = 2$.

At Step 2, we include in C an internal vertex, which mirrors the singular vertex labeled a in V, with 1 root, since there is a single sprout in U. Accordingly, we define $\xi_U(x) = 1$. □

Example 12. Fig. 14 shows how cycles may result from unifying non-linear drags. The congruence obtained by unifying the input drags at the pair of roots labeled by f has 3 classes:

$(U[f \cdot \mathbf{1}], V[f \cdot \mathbf{1}]$
$(U[x \cdot \mathbf{2} \cdot \mathbf{4}], U[x \cdot \mathbf{4}], V[h \cdot \mathbf{2}])$
$(U[h \cdot \mathbf{3}], V[y \cdot \mathbf{5}]V[y \cdot \mathbf{3} \cdot \mathbf{5}])$

At step 1, corresponding to the rooted internal vertices in $U \oplus V$, sprouts x_1, x_2 are included in C and sprouts, y_1, y_2 in D, all of them with 1 root each. Accordingly, $\xi_C(x_1) = 1; \xi_C(x_2) = 2; \zeta_D(y_1) = 1; \zeta_D(y_2) = 2$;

At step two, we include in C a vertex labeled h, mirroring vertex $V[h \cdot \mathbf{2})$ in V, with three roots, since $V[h \cdot \mathbf{2})$ has one root and there are two sprouts from U in the corresponding equivalence class. Similarly, we include in D a vertex labeled h, mirroring vertex $V[h \cdot \mathbf{3}]$, with three roots. In this case, ξ_U sends both x's to the two roots of the vertex labeled h in C and ζ_V sends both x's to the two roots of the vertex labeled h in D.

Finally, at step 3, the successor of the vertex labeled h in C is defined to be x_2 and the successor of the vertex labeled h in D is defined to be y_2.

Note that, in the figure, the lefthand side h vertex of the unified drag is the mirror of the h vertex of the righthand side input drag in the left overlap, while the righthand side h vertex is the mirror vertex of the h vertex of the lefthand side input drag in the right overlap. So, both overlaps are not really identical as drags, although their drawing is the same. □

We now prove that (C, ξ) and (D, ζ) are rewriting extensions and a solution of the given unification problem.

Lemma 15. *Let $\equiv$ be an equivalence in solved form for the unification problem $U[\overline{u}] = V[\overline{v}]$. Then, the most general unifying extensions $\langle C, \xi \rangle$ and $\langle D, \zeta \rangle$ is a solution of the unification problem.*

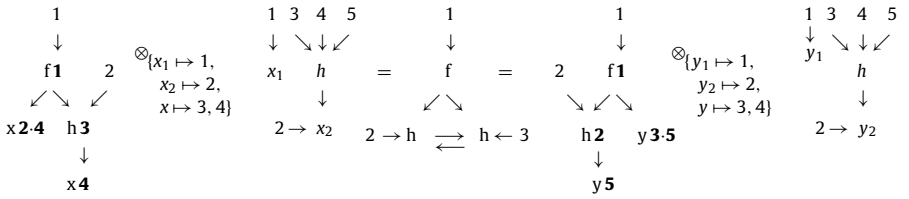


Fig. 14. Building a most general unifier from a solved congruence.

Proof. We show first that $\langle C, \xi\rangle$ and $\langle D, \zeta\rangle$ are rewriting extensions. We carry out the proof for $\langle C, \xi\rangle$, the other being similar.

- First, the switchboard ξ is clearly well-defined.
- Totality of ξ_U: each sprout s of U belongs to some class S, hence is mapped by ξ to s_S.
- Surjectivity and linearity of ξ_C: by construction, for every internal rooted vertex, say u, belonging to a class S, there exists s_S in C such that $\xi(s_S) = u$. And for every rooted sprout r in U, there is a sprout s_r in C such that $\xi(s_r) = r$. Finally, C is linear by construction.
- Cleanliness: let u be a vertex in $C \otimes_\xi U$. By totality, it can't be a sprout of U.
  If u is an internal vertex of C, then none of its ancestors can be vertices of U, hence all are mirror vertices of vertices in V with the same number of roots by construction, which ensures that u is accessible in $C \otimes_\xi U$.
  If u is an internal vertex of U, then u is accessible in U, hence in $C \otimes U$ from some vertex u' which is rooted in U. If u' is rooted in $\otimes_\xi U$, we are done. Otherwise, u' must have a predecessor in C, which is accessible, as we have already proved.
  If u is a sprout of C, then u is a fresh sprout s_S, the equivalence class S being a set of sprouts of U, V. In that case, depending whether all sprouts are on the V side or not, u has an ancestor in either U or C, which must be accessible by the two previous cases, hence u is accessible.

Let us show that $U' = C \otimes_\xi U =^\iota D \otimes_\zeta V = V'$:

- By definition, $\mathcal{I}nt(C \otimes_\xi U) = \mathcal{I}nt(U) \cup \mathcal{I}nt(C) = \{u \in \mathcal{I}nt(U) \mid u \text{ is not singular}\} \cup \{u \in \mathcal{I}nt(U) \mid u \text{ is singular}\} \cup \{v_S \mid v \in \mathcal{I}nt(V) \text{ and } v \text{ is singular}\}$. Then, $\iota(\mathcal{I}nt(C \otimes_\xi U)) = \iota(\{u \in \mathcal{I}nt(U) \mid u \text{ is not singular}\}) \cup \iota(\{u \in \mathcal{I}nt(U) \mid u \text{ is singular}\}) \cup \iota(\{v_S \mid v \in \mathcal{I}nt(V) \text{ and } v \text{ is singular}\}) = \mathcal{I}nt(D \otimes_\zeta V)$;
- By definition, $\mathcal{S}(C \otimes_\xi U) = \{s_S \mid S \cap (\mathcal{I}nt(U) \cup \mathcal{I}nt(V)) = \varnothing\}$. Then, $\iota(\mathcal{S}(C \otimes_\xi U)) = \iota(\{s_S \mid S \cap (\mathcal{I}nt(U) \cup \mathcal{I}nt(V)) = \varnothing\}) = \mathcal{S}(D \otimes_\zeta V)$
- By definition the labels of internal vertices in $C \otimes_\xi U$ and $D \otimes_\zeta V$ coincide. Moreover, for each pair of sprouts s_S and t_S, their label is x_S in both drags (having x_S and $y_S \neq x_S$ instead would require an additional variable renaming to show that both drags are equal modulo renaming).
- For each $w \in \mathcal{I}nt(C \otimes_\xi U)$, if $X_{C \otimes_\xi U}(w) = \langle w_1 \dots w_k\rangle$, then $X_{D \otimes_\zeta V}(\iota(w)) = \langle \iota(w_1) \dots \iota(w_k)\rangle$, since $\equiv$ is a congruence.
- Finally, the number of roots at each vertex u in $C \otimes_\xi U$ is equal to the number of roots at $\iota(u)$ in $D \otimes_\zeta V$: If s_S is a sprout in $C \otimes_\xi U$, then $\iota(s_S) = t_S$ and both sprouts have one root in $C \otimes_\xi U$ and $D \otimes_\zeta V$, by definition. If u is an internal vertex in U, whose equivalence class S does not include any internal vertex, by definition, u and $\iota(u)$ will have the same number of roots in $C \otimes_\xi U$ and $D \otimes_\zeta V$. Finally, if S includes internal vertices u, v, with $\iota(u) = v$, then, if both u and v are rooted in U and in V then, by definition, u and v will be rooted in $C \otimes_\xi U$ and $D \otimes_\zeta V$. But if one of the vertices u or v is unrooted in U or V, respectively then, both vertices will be unrooted in $C \otimes_\xi U$ and $D \otimes_\zeta V$. □

As an important consequence, we have

Corollary 1. *Unification congruence $=_{unif}$ and marking congruence $\equiv_{unif}$ coincide.*

Proof. Let $U[\overline{u}] = V[\overline{v}]$ be a unification problem. The result is clear if it is unsolvable. Otherwise, let $\equiv$ be a congruence in solved form for that problem. By Lemma 15, $\equiv$ is the equivalence associated with the unifying extensions introduced at Definition 23, hence is coarser than $=_{unif}$ which is the intersection of all equivalences associated with the solutions of a given unification problem. Now, it is easy to see that $=_{unif}$ is itself a congruence in solved form, hence is coarser than $\equiv$ by Lemma 14. □

3.7. Completeness of the unification algorithm

Theorem 1. *Let $\equiv$ be the equivalence returned by the unification algorithm for the problem $U[\overline{u}] = V[\overline{v}]$ when no failure occurs. Then, $mgu(U \oplus V, \equiv)$ is a most general unifier.*

Proof. By Lemma 15, $(\langle C, \xi\rangle, \langle D, \zeta\rangle) = mgu(U[\overline{u}] \oplus V[\overline{v}], \equiv)$ is a unifier of the equation $U[\overline{u}] = V[\overline{v}]$. We are therefore left proving it is most general.

Let now S be a solution of $U[\overline{u}] = V[\overline{v}]$. By Corollary 1, S defines a unification equivalence whose classes must be (appropriate) unions of classes of $\equiv$. It therefore suffices to show that merging classes of a congruence in solved form increases (w.r.t. subsumption) the associated solution constructed at Lemma 15.

We describe the situation when merging two mergeable classes S and S'. There are four cases up to symmetry depending upon the existence of internal vertices in these classes, since there can only be at most two internal vertices on opposite sides in a merged class. We carry out one case, with two internal vertices in S and none in S', the others being similar.

Let therefore $S = \{u, s_1, \ldots s_m, t_1, \ldots, t_n, v\}$ and $S' = \{s'_1, \ldots, s'_{m'}, t'_1, \ldots t'_{n'}\}$. Lemma 15 creates the sprouts $s_S, t_S, s_{S'}, t_{S'}$, as well as additional rootless sprouts to annihilate the roots of the rooted sprouts of S and S'. After merging the two classes, Lemma 15 uses new sprouts s_m and t_m, whose number of roots is just the sum of those for s_S and $s_{S'}$, while the additional sprouts can remain the same. The latter solution is therefore an instance of the previous one via the following extension: two new sprouts s, s' equipped with two roots each. Then the extension's switchboard can map s_S and $s_{S'}$ to s_m and t_S and $t_{S'}$ to t_m, this will give a solution for the merged class.

Merging more than two classes obeys the same mechanism. Further, Lemma 2 ensures that merges can be done one by one. □

3.8. 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 7 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. □

The question therefore arises whether these complexity bounds are sharp. A long way was necessary to go from the early unification algorithm of terms by Robinson, which was exponential, to the linear time unification algorithm of Patterson and Wegman. It is indeed easy to go from quadratic to linear space by changing our marking technique, since a single number per vertex is enough for memorizing an equivalence class, instead of a list of numbers of quadratic length. Going from quadratic to sub-quadratic time complexity should be harder, and might require a smart strategy for applying the unification rules, as is the case for terms [25]. Whether the ultimate algorithm should be linear or not is an open problem for which an early guess would be premature. On the other hand, it might be easy to use Tarjan's union-find algorithm as done by Huet for trees [22], and therefore obtain a quasi-linear algorithm for drag unification.

4. Confluence

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

Definition 24 *(Local peaks).* A *local peak* is a pair of rewrites $S \longleftarrow^{\mu}_{L\to R} U \longrightarrow^{\nu}_{G\to D} T$ originating from a given drag U. A local peak is *overlapping* or *critical* if there exist internal vertices u of L, v of G and w of U such that $\mu(u) = \nu(v) = w \in \mathcal{I}nt(U)$, in which case we say that L and G *share* the internal vertex w of U. A local peak is *disjoint* otherwise.

In the case of terms, non-overlapping peaks are either disjoint or ancestors, depending on their relative positions, and both join. Here, there is no distinction between non-overlapping peaks, all are disjoint, a definition which covers both kinds of peaks for terms. This is the benefit of our definition of composition of drags, which subsumes both operations of context application and substitution for terms. Further, the coming commutation proof happens to be simpler, since non-linearities of variables are taken care of by the composition mechanism. It is just like the case of disjoint redexes for terms, the reason why we retain the name.

Lemma 16 *(Commutation). Let $S \longleftarrow^{\mu'}_{L\to R} U \longrightarrow^{\nu'}_{G\to D} T$ be a disjoint local peak. Then, $S \longrightarrow_{G\to D} V \longleftarrow_{L\to R} T$ for some drag V.*

Proof. W.l.o.g., we assume that all vertices of U are internal. By definition of rewriting, there exist rewriting extensions $\langle C, \mu\rangle$ of L at μ' and $\langle D, \nu\rangle$ of G at ν' so that $C \otimes_{\mu} L = U = D \otimes_{\nu} G$.

By the disjointness assumption on the local peak, there exist no pair of vertices $u \in \mathcal{I}nt(L), v \in \mathcal{I}nt(G)$ such that $\mu'(u) = \nu'(u) = w \in \mathcal{I}nt(U)$. It follows that $\nu'(\mathcal{I}nt(G)) \subseteq \mathcal{I}nt(C)$ and $\mu'(\mathcal{I}nt(L)) \subseteq \mathcal{I}nt(D)$. This allows to exhibit a new map $\theta' : G \to C$ which coincides with ν' on $\mathcal{I}nt(G)$:

(i) $\theta'(u) = \nu'(u)$ for all $u \in \mathcal{V}er(G)$ such that $\nu'(u) \in \mathcal{I}nt(C)$;

(ii) let now $s : x$ be a sprout of G such that $\nu'(s) = v \notin \mathcal{I}nt(C)$, hence $v \in \mu'(\mathcal{I}nt(L))$ by definition of composition. Since there are no isolated sprouts in patterns, there must exist an internal vertex w in G such that s is the i-th successor of w in G. This implies that $\nu'(w) \in \mathcal{I}nt(C)$, and v is the i-th successor of $\nu'(w)$ in U. By definition of composition, since $v \in \mu'(\mathcal{I}nt(L))$, there must exist a sprout t_v in C such that t_v is the i-th successor of $\nu'(w)$ in C and $\mu_C(t_v) = v$, implying that v is rooted. We then define $\theta'(s) = t_v$.

We claim that the map θ' is an injection: (i) it clearly preserves successorship, and (ii) θ' sends sprouts labeled by the same variable to the same vertex of C since ν' forces sharing, and (iii) the new edge property is satisfied since it is either inherited from ν' in case (i), and results from the fact that v is rooted in case (ii).

By Lemma 4, there exists an extension $\langle E, \theta\rangle$ of G such that $C = E \otimes_\theta G$, hence $U = (E \otimes_\theta G) \otimes_\xi L$. By Lemma 2, $U = E \otimes_\gamma (G \otimes_\eta L)$, where γ, η are clean rewriting extensions, hence $S = E \otimes_\gamma (G \otimes_\eta R)$ and $T = E \otimes_\gamma (D \otimes_\eta L)$, since the result of rewriting at some position is unique, as a consequence of Lemma 4.

Then $V = E \otimes_\gamma (D \otimes_\eta R)$ satisfies the claim thanks to Lemma 5. □

Lemma 17 *(Critical peak). Let $S \longleftarrow^{\mu'}_{L\to R} U \longrightarrow^{\nu'}_{G\to D} T$ be a critical peak such that $\langle C, \mu\rangle$ and $\langle E, \nu\rangle$ are the rewriting extensions of L at μ' and G at ν' respectively. Then, there exist vertices $\overline{u} \in \mathcal{V}er(L)$ and $\overline{v} \in \mathcal{V}er(G))$ such that the extensions $\langle C, \mu\rangle$, $\langle E, \nu\rangle)$ form a solution to the unification problem $U[\overline{u}] = V[\overline{v}]$..*

Proof. Let A be the subset of internal vertices of U belonging to both $\mu'(\mathcal{I}nt(L))$ and $\nu'(\mathcal{I}nt(G))$. By assumption, $A \neq \varnothing$, implying that L and G overlap. The core of the proof is the definition of two lists (or sets) $\overline{v}, \overline{w}$ of partner vertices of L, G that generate A via μ' and ν', that is, such that all vertices of A are accessible in U from both $\mu'(\overline{v})$ and $\nu'(\overline{w})$, and vertices of L, G that are not reachable from $\overline{v}$ and $\overline{w}$ must belong to the contexts C, E. As partner vertices, v_i and w_i must coincide, that is, be mapped by μ' and ν' to a same internal vertex of A.

Denoting by X_A the successor relationship restriction of X_U to A, we say that a vertex u in A is minimal if any ancestor v of u in A, that is $v\, X_A^*\, u$, is also a successor of u, that is $u\, X_A^*\, v$. We then consider a non-redundant set B of minimal vertices (of A) generating A. By its definition, any vertex in A is the successor of a vertex in B. Since any vertex in A is the image of both an internal vertex of L by μ' and an internal vertex of G by ν', both injective on internal vertices, we define the set of partner vertices as $P = \{(v, w) : \mu'(v) = \nu'(w) \in B\}$.

We now show that the extensions $\langle C, \mu\rangle$, $\langle E, \nu\rangle)$ form a solution to the unification problem $U[\overline{u}] = V[\overline{v}]$. By definition of rewriting, there exist renamings L' and G' of L and G respectively such that $U = L' \otimes_\mu C = G' \otimes_\nu E$ for some vertex renamings L' of L and G' of G. By Lemma 1, $L \otimes_\mu C =^\iota G \otimes_\nu E$ for some vertex renaming ι. $w = \iota(v)$ for each pair of partner vertices, (v, w) since both are sent to the same vertex of U by μ and μ'. Finally, by definition of A, true ancestors of $\overline{v}$ must belong to the context D (and true ancestors of $\overline{w}$ to C).

Hence L and G are unifiable at P. □

Definition 25 *(Critical pair, joinability).* Let $L \to R$ and $G \to D$ be two rules whose left-hand sides L and G are unifiable at partner vertices $P = (\overline{v}, \overline{w})$, with most general solution $(\langle C, \mu\rangle, \langle E, \nu\rangle)$. Then, $\langle C \otimes_\mu R, E \otimes_\nu D\rangle$ is a *critical pair* of $L \to R$ with $G \to D$ at P, and $C \otimes_\mu L$ (equivalently, $D \otimes_\nu G$) a *critical overlap* of L, G below P.

A critical pair $\langle C_1, C_2\rangle$ is said to be *joinable* if there is a drag C_3 such that $C_1 \longrightarrow^* C_3$ and $C_2 \longrightarrow^* C_3$.

Given a drag rewriting system, how many critical pairs can be generated? Their number is indeed bounded by the potential choices for partner vertices, a non-redundant set of pairs of vertices labeled by the same function symbol. Despite the potential exponential number of them, there should be relatively few possibilities in practice, as is the case for terms.

We will now present our second main result:

Theorem 3. *Let $\mathcal{R}$ be a rewrite system on drags. Then, $\mathcal{R}$ 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 Lemma 16.
2. G has a vertex in common with L. By Lemma 17, there is a critical pair. By property of most general unifiers and Lemma 5, 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 [15,16,28]. 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 26. A drag rewriting system is said to be *orthogonal* if it does not possess critical pairs.

Note that left-linearity is not needed here: a non-left linear rule and the linear rule obtained by sharing all sprouts labeled by the same variable define the same rewriting relation on terms. Hence our definition of drag rewriting is inherently linear, as we have remarked already.

Theorem 4. *Orthogonal drag rewriting systems are confluent.*

Proof. Lemma 16 shows that rewriting has the so-called diamond under the assumption of orthogonality, hence can be shown confluent by the standard pasting technique. □

We are currently developing a new version of drags for which linearity is not built in the definition of composition as it is here. This new model would require the assumption of left-linearity for orthogonal systems.

5. Related work

The first, dominant model for graph rewriting was introduced in the mid-seventies by Hartmut Ehrig, Michael Pfender and Jürgen Schneider [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 concrete graphs, namely those that do not admit cycles [35]. In particular, termination and confluence techniques have been elaborated for various generalizations of trees, such as rational trees, directed acyclic graphs, jungles, term-graphs, lambda-graphs, as well as for graphs in general. See [19] for a survey of various forms of graph rewriting and of available analysis techniques.

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, except in the case of symbolic graphs [29], where variables may just denote values. One might argue that the first drawback has not completely disappeared with drags: a left-hand side of rule may match a drag D with a switchboard which is ill-behaved with respect to the right-hand side R of the rule, hence forbidding its application. However, this can only happen in a very restricted case: D must contain a loop on a vertex labeled f, and the rule must be of the form $f(\ldots, x, \ldots) \rightarrow x$ with one root on each side, the variable x matching the loop of D.

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 [13,36]), or Sesqui-Pushout transformation (SqPO [6]), AGREE [5], and *Hyperedge Replacement Systems* [11]. A detailed comparison of the approach based on drags with all these approaches is not obvious and is carried out in [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 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. An old work that has similarities with drag rewriting, in particular the objective of generalizing term rewriting in a natural way, is the hypergraph model of Bauderon and Courcelle [2]. Like drags, it has symbols with arities as well as a list of roots called sources there. It has also an algebraic theory based on the same sum operation as well as operations on sources which are quite different from our composition operator, since there is no notion of variable in their model. Rewriting is done by exhibiting an injective morphism first, and then using gluing for constructing the right-hand side, in a way which resembles DPO very much. A recent approach that has also some similarities with drag rewriting 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 [3].

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., [31]). A preliminary attempt to handling variables in graph unification is [30], where variables are used to represent labels equipping the vertices or edges. A quite more general approach is [35], 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 [29]. 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. This major result 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 labeling 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 [32–34]. He also considered the case of symbolic graphs [20]. A main problem with Plump's notion of critical pair is that they are too many. 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 L_2 and 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. Less prolific notions of critical pairs have been introduced in [1,26,27]. For instance, [26] 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 left-hand side of the rule with itself was 51602; the number of critical pairs according to Plumps definition was 21478; the number of critical pairs computed using the method presented in [27] was 49; finally, the number of critical pairs computed using the method presented in [1,26] was 7.

Recently, local confluence was shown decidable for terminating graph rewriting with interfaces [3], 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 [3], means rewriting the graph but leaving the interface invariant: the interface restricts the application of rules, since it must be preserved. 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 [3]:

Consider the following rewrite rules on directed graphs with labeled edges, where $\xrightarrow{a}$ represents an edge labeled by a and vertices are subindexed with numbers 1 and 2 to identify them and make the morphisms explicit:

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

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

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

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

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

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

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:

$$\begin{array}{ccccccc} & [\bullet_1 \circlearrowright & \bullet_2] & \longleftarrow & [\bullet_1 \xrightarrow{a} \bullet_1] & \longrightarrow & [\bullet_1 \quad \bullet_2 \circlearrowright] \\ [\bullet_1 \quad \bullet_2] & & [\bullet_1 \quad \bullet_2] & & [\bullet_1 \quad \bullet_2] & & \end{array}$$

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, hence confluence, of terminating systems on all terms is decidable while it becomes undecidable on ground terms.

In the case of drags, the analogy to term rewriting is more faithful, because drags generalize terms. Confluence of terminating drag rewriting on ground expressions is therefore not decidable, but we have shown here that it is decidable for arbitrary drags with variables. Furthermore, as with term rewriting, drag critical pairs are computed via most general unifiers and all of them are needed. The reason why their number is limited, as we already pointed out, 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. This would of course require generalizing our unification algorithm as well as our confluence result itself to account for commutative, or associative commutative symbols. We know by experience with the case of terms that neither one is simple [24,38].

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 is 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. We do not forget the referees of a previous submission, who forced us to make precise the mathematics of this paper, in particular to define clearly drag equality and drag matching used throughout the paper as well as the key new notion of unifying extensions.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

References

[1] Guilherme Grochau Azzi, Andrea Corradini, Leila Ribeiro, On the essence and initiality of conflicts in m-adhesive transformation systems, J. Log. Algebraic Methods Program. 109 (2019).
[2] Michel Bauderon, Bruno Courcelle, Graph expressions and graph rewritings, Math. Syst. Theory 20 (2–3) (1987) 83–127.
[3] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, Fabio Zanasi, Confluence of graph rewriting with interfaces, in: Hongseok Yang (Ed.), 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, in: Lecture Notes in Computer Science, Springer, 2017, pp. 141–169.
[4] Horatiu Cirstea, David Sabel (Eds.), Proceedings Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2017, Oxford, UK, September 2017, EPTCS, vol. 265, 2018.
[5] Andrea Corradini, Dominique Duval, Rachid Echahed, Frédéric Prost, 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, 2015, pp. 35–51.
[6] Andrea Corradini, Tobias Heindel, Frank Hermann, Barbara König, Sesqui-pushout rewriting, in: Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17–23, 2006, Proceedings, 2006, pp. 30–45.
[7] Nachum Dershowitz, Jean-Pierre Jouannaud, Rewrite systems, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier, 1990, pp. 243–320.

[8] Nachum Dershowitz, Jean-Pierre Jouannaud, Graph path orderings, in: Gilles Barthe, Geoff Sutcliffe, Margus Veanes (Eds.), LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, in: EPiC Series in Computing, vol. 57, EasyChair, 2018, pp. 307–325.
[9] Nachum Dershowitz, Jean-Pierre Jouannaud, Drags: a compositional algebraic framework for graph rewriting, Theor. Comput. Sci. 777 (2019) 204–231.
[10] Nachum Dershowitz, Jean-Pierre Jouannaud, Fernando Orejas, Graph rewriting cum drag rewriting, Work in preparation.
[11] Frank Drewes, Hans-Jörg Kreowski, Annegret Habel, Hyperedge replacement, graph grammars, in: Rozenberg [37], pp. 95–162.
[12] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, Gabriele Taentzer, Fundamentals of Algebraic Graph Transformation, Springer-Verlag, 2006.
[13] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, Andrea Corradini, Algebraic approaches to graph transformation – part II: single pushout approach and comparison with double pushout approach, in: Rozenberg [37], pp. 247–312.
[14] Hartmut Ehrig, Michael Pfender, Hans Jürgen Schneider Graph-grammars, An algebraic approach, in: 14th Annual Symposium on Switching and Automata Theory, Iowa City, IA, IEEE Computer Society, October 1973, pp. 167–180.
[15] Bertram Felgenhauer, Rule labeling for confluence of left-linear term rewrite systems, in: IWC, 2013, pp. 23–27.
[16] Bertram Felgenhauer, Aart Middeldorp, Harald Zankl, Vincent van Oostrom, Layer systems for proving confluence, ACM Trans. Comput. Log. 16 (2) (2015) 14:1–14:32.
[17] Maribel Fernández, Hélène Kirchner, Bruno Pinaud, Strategic port graph rewriting: an interactive modelling framework, Math. Struct. Comput. Sci. 29 (5) (2019) 615–662.
[18] Annegret Habel, Hans-Jörg Kreowski, Detlef Plump, Jungle evaluation, Fundam. Inform. 15 (1) (1991) 37–60.
[19] Reiko Heckel, Gabriele Taentzer (Eds.), Graph Transformation, Specifications, and Nets – In Memory of Hartmut Ehrig, Lecture Notes in Computer Science, vol. 10800, Springer, 2018.
[20] Ivaylo Hristakiev, Detlef Plump, Checking graph programs for confluence, in: Martina Seidl, Steffen Zschaler (Eds.), Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops, Marburg, Germany, July 17-21, 2017, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 10748, pp. 92–108, 2017.
[21] Ivaylo Hristakiev, Detlef Plump, A unification algorithm for GP 2 (long version), CoRR, arXiv:1705.02171 [abs], 2017.
[22] Gérard Huet, Unification dans les langages d'ordre 1, ..., ω, PhD thesis, Université Paris 7, Paris, France, 1976.
[23] Gérard P. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, J. ACM 27 (4) (1980) 797–821.
[24] J.-P. Jouannaud, H. Kirchner, Completion of a set of rules modulo a set of equations, SIAM J. Comput. 15 (4) (1986) 1155–1194.
[25] Jean-Pierre Jouannaud, Claude Kirchner, Solving equations in abstract algebras: a rule-based survey of unification, in: Jean-Louis Lassez, Gordon D. Plotkin (Eds.), Computational Logic - Essays in Honor of Alan Robinson, The MIT Press, 1991, pp. 257–321.
[26] Leen Lambers, Kristopher Born, Fernando Orejas, Daniel Strüber, Gabriele Taentzer, Initial conflicts and dependencies: critical pairs revisited, in: Graph Transformation, Specifications, and Nets - in Memory of Hartmut Ehrig, 2018, pp. 105–123.
[27] Leen Lambers, Hartmut Ehrig, Fernando Orejas, Efficient conflict detection in graph transformation systems by essential critical pairs, Electron. Notes Theor. Comput. Sci. 211 (2008) 17–26.
[28] Jiaxiang Liu, Jean-Pierre Jouannaud, Mizuhito Ogawa, Confluence of layered rewrite systems, in: Stephan Kreutzer (Ed.), 24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany, in: LIPIcs, vol. 41, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 423–440.
[29] Fernando Orejas, Leen Lambers, Symbolic attributed graphs for attributed graph transformation, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 30 (2010).
[30] Francesco Parisi-Presicce, Hartmut Ehrig, 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, 1986, pp. 496–514.
[31] Mike Paterson, Mark N. Wegman, Linear unification, J. Comput. Syst. Sci. 16 (2) (1978) 158–167.
[32] Detlef Plump, Hypergraph rewriting: critical pairs and undecidability of confluence, in: Ronan Sleep, Rinus Plasmeijer, Marko van Eekelen (Eds.), Term Graph Rewriting: Theory and Practice, John Wiley, 1993, pp. 201–213.
[33] Detlef Plump, Critical pairs in term graph rewriting, in: Igor Prívara, Branislav Rovan, Peter Ruzicka (Eds.), Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science (MFCS '94), Kosice, Slovakia, in: Lecture Notes in Computer Science, vol. 841, Springer, August 1994.
[34] Detlef Plump, Confluence of graph transformation revisited, in: Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, Roel C. de Vrijer (Eds.), Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 3838, Springer, 2005, pp. 280–308.
[35] Detlef Plump, Annegret Habel, Graph unification and matching, in: Janice E. Cuny, Hartmut Ehrig, Gregor Engels, Grzegorz Rozenberg (Eds.), Selected Papers of the 5th International Workshop on Graph Grammars and Their Application to Computer Science, in: Lecture Notes in Computer Science, vol. 1073, Springer, Williamsburg, VA, November 1994, pp. 75–88.
[36] Raoult Jean-Claude, On graph rewritings, Theor. Comput. Sci. 32 (1984) 1–24.
[37] Grzegorz Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, World Scientific, 1997.
[38] Mark E. Stickel, A unification algorithm for associative-commutative functions, J. ACM 28 (3) (1981) 423–434.