

Graph Unification and Matching*

Detlef Plump
Universität Bremen[†]

Annegret Habel
Universität Hildesheim[‡]

Abstract

A concept of graph unification and matching is introduced by using hyperedges as graph variables and hyperedge replacement as substitution mechanism. It is shown that a restricted form of graph unification corresponds to solving linear Diophantine equations, and hence is decidable. For graph matching, transformation rules are given which compute all (pure) solutions to a matching problem. The matching concept suggests a new graph rewriting approach which is very simple to describe and which generalizes the well-known double-pushout approach.

1 Introduction

The unification of expressions plays an important role in many areas of computer science, for example in theorem proving, logic programming, type inference, and natural language processing (see the surveys on unification of Knight [15], Jouanaud and Kirchner [14], and Baader and Siekmann [1]). While expressions are usually considered as strings or trees, efficient unification algorithms often use graph representations of expressions in order to exploit the sharing of common subexpressions. This applies, for example, to the algorithms of Huet [11], Paterson and Wegman [18], Corbin and Bidoit [6], and Jaffar [12]. Thus it seems natural to abstract from implementation details and to study a concept of *graph unification* in its own right.

With a somewhat different motivation, Parisi-Presicce, Ehrig, and Montanari [17] consider graph rewriting with unification. In their approach, variables are instantiated by graph morphisms and graph rewriting is defined by double-pushouts of graph morphisms. In the introduction of [17], the authors identify the following problem: “One of the basic concepts missing is a suitable notion of variable and of graph unification.” We also address this problem and propose

*Work partially supported by ESPRIT Basic Research Working Group COMPUGRAPH II, # 7183, and by Deutsche Forschungsgemeinschaft, # Kr 964/2-2.

[†]Address: Fachbereich Mathematik und Informatik, Universität Bremen, Postfach 33 04 40, 28334 Bremen, Germany. E-mail: `det@informatik.uni-bremen.de`.

[‡]Address: Institut für Informatik, Universität Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany. E-mail: `habel@informatik.uni-hildesheim.de`.

a solution which we consider as natural and simple: *hyperedges* serve as graph variables and *hyperedge replacement* is used to substitute graphs for variables.

In this paper, we study this idea in its pure form, without tailoring graph unification to the implementation of term unification. The latter requires both restriction to a smaller graph class (term graphs) and consideration of a graph equivalence more liberal than isomorphism. A corresponding adaptation of the present approach can be found in [10].

Although the general graph unification problem is easy to state (determine, for any hypergraphs G and H , whether there is a substitution making G and H isomorphic), we do not know whether the problem is decidable or not. Below we solve a restricted form of the graph unification problem, solve the general graph matching problem (unification in the case where one graph is variable-free), and define a substitution-based graph rewriting approach:

- In the restricted case where variables are not attached to nodes, the graph unification problem is equivalent to the problem of finding non-negative integer solutions to a set of linear Diophantine equations. This in turn is known to be equivalent to the restricted AC1 unification problem for first-order terms, where terms are built up from an associative-commutative function symbol, a unit element, and free constants.
- There is a graph matching algorithm based on transformation rules which computes all solutions (without useless variables) to a given matching problem.
- Substitution-based graph rewriting is introduced. In this approach, rules are just pairs $\langle L, R \rangle$ of hypergraphs which are applied by matching L to a given hypergraph and by applying the corresponding substitution to R . This approach allows to copy and delete non-local subgraphs, and generalizes the well-known double-pushout approach.

2 Hypergraphs and substitutions

Let Σ be a set the elements of which are called *labels*. Each label l comes with a natural number $type(l) \geq 0$. A *hypergraph* over Σ is a system $G = \langle V_G, E_G, lab_G, att_G \rangle$ consisting of two finite sets V_G and E_G of *nodes* and *hyperedges*, a labelling function $lab_G: E_G \rightarrow \Sigma$, and an attachment function $att_G: E_G \rightarrow V_G^*$ which assigns a string of nodes to a hyperedge e such that $type(lab_G(e))$ is the length of $att_G(e)$.

A *hypergraph morphism* $f: G \rightarrow H$ consists of two functions $f_V: V_G \rightarrow V_H$ and $f_E: E_G \rightarrow E_H$ that preserve labels and attachment to nodes, that is, $lab_H \circ f_V = lab_G$ and $att_H \circ f_E = f_V^* \circ att_G$. (The extension $f^*: A^* \rightarrow B^*$ of a function $f: A \rightarrow B$ maps the empty string to itself and $a_1 \dots a_n$ to $f(a_1) \dots f(a_n)$.) The morphism f is *injective* (*surjective*) if f_V and f_E are injective (surjective), and is an *isomorphism* if it is injective and surjective. In the latter case, G and H are *isomorphic*, which is denoted by $G \cong H$.

A *pointed hypergraph* $\langle G, p_G \rangle$ is a hypergraph G together with a sequence $p_G = v_1 \dots v_n$ of pairwise distinct nodes from G . We write $type(G)$ for the number n of points and P_G for the set $\{v_1, \dots, v_n\}$. If convenient, we denote a pointed hypergraph only by its hypergraph component.

We assume that there is a distinguished subset Var of Σ the elements of which are used as *variables* and which are denoted by x, y, z, x_1, x_2, \dots . The set of variables occurring in a hypergraph G is denoted by $Var(G)$, and we write G^\ominus for the hypergraph obtained from G by removing all hyperedges labelled with variables.

Definition 1 A *substitution pair* x/G consists of a variable x and a pointed hypergraph G such that $type(x) = type(G)$. A *substitution* is a finite set $\sigma = \{x_1/G_1, \dots, x_n/G_n\}$ of substitution pairs such that x_1, \dots, x_n are pairwise distinct. The *domain* of σ is the set $Dom(\sigma) = \{x_1, \dots, x_n\}$.

Given a hypergraph H and a hyperedge e in H labelled with x , the application of a substitution pair x/G to e proceeds in two steps: (1) Remove e from H , yielding the hypergraph $H - \{e\}$, and (2) construct the disjoint union $(H - \{e\}) + G$ and fuse the i^{th} node in $att_H(e)$ with the i^{th} point of G , for $i = 1, \dots, type(x)$. (Note that if $att_H(e)$ contains repeated nodes, then the corresponding points in G are identified.)

The application of a substitution σ to a hypergraph H yields the hypergraph $H\sigma$ which is obtained by applying all substitution pairs in σ simultaneously to all hyperedges with label in $Dom(\sigma)$.

Given two substitutions σ and τ , the *composition* $\sigma\tau$ is the substitution $\{x/G\tau \mid x/G \in \sigma\} \cup \{y/H \in \tau \mid y \notin Dom(\sigma)\}$. By associativity of hyperedge replacement (see [9]), the composition has the following important property.

Lemma 2 For all hypergraphs G and substitutions σ and τ , $G(\sigma\tau) \cong (G\sigma)\tau$.

3 Unification

After a notion of substitution is introduced, it is obvious how to define unification and matching. In this section we show that graph unification is decidable for a restricted class of problems, and we give sufficient conditions for the existence of solutions in the general case.

Definition 3 A *unification problem* $G =? H$ consists of two hypergraphs G and H . A *solution* to this problem is a substitution σ such that $G\sigma$ and $H\sigma$ are isomorphic.

Example 4 The unification problem in Figure 1 consists of two flow graphs with variables S and B (standing for statements and boolean expressions). The rectangles represent hyperedges of type 2 which are connected with their first and second attachment node by a line and an arrow, respectively. Rhombi represent hyperedges of type 3, where arrows labelled with t and f point to the second and

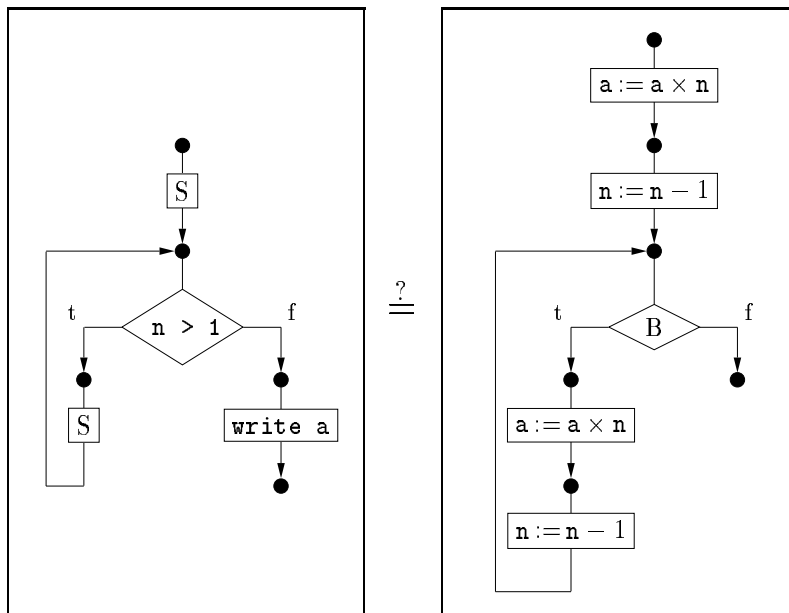


Figure 1: A unification problem

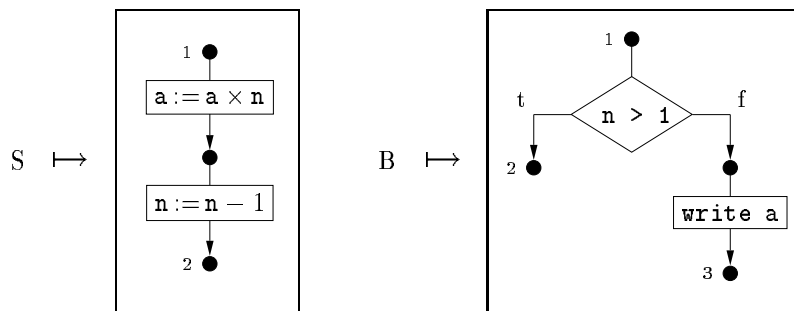


Figure 2: A solution to the problem in Figure 1

third attachment node, respectively. A solution to this unification problem is the substitution shown in Figure 2 (where the numbered nodes are the points of the hypergraphs). Another solution is obtained if the node above the hyperedge labelled with “**write a**” becomes the third point in the right hypergraph.

The *general unification problem* is the problem to decide, for any unification problem $G =? H$, whether it has a solution or not. Note that for hypergraphs without variables, the general unification problem is the hypergraph isomorphism problem.

We do not know whether the general unification problem is decidable or not. Below we establish decidability for the case that all variables have type 0, that is, for the case that variable hyperedges are not attached to any nodes. Even this strongly restricted kind of unification turns out to be non-trivial. Essentially, here unification is equivalent to solving a system of linear Diophantine equations.

Definition 5 The *type* of a unification problem $G =? H$ is the maximal type of the variables in G and H . If there are no variables, then the type is \perp .

Consider a unification problem $G =? H$ of type 0. Let C_1, \dots, C_m be the non-isomorphic, connected, variable-free components and x_1, \dots, x_k be the variables occurring in G and H . We represent G and H as disjoint unions of their connected components: $G = \sum_{i=1}^k a_i X_i + \sum_{j=1}^m b_j C_j$ and $H = \sum_{i=1}^k a'_i X_i + \sum_{j=1}^m b'_j C_j$, where a_i, a'_i, b_j, b'_j are non-negative integers and X_i stands for a hyperedge labelled with x_i . Now assign to G and H the following set $Eq(G =? H)$ of linear Diophantine equations:

$$Eq(G =? H) = \left\{ \sum_{i=1}^k (a_i - a'_i) x_{i,j} = (b'_j - b_j) \mid 1 \leq j \leq m \right\}.$$

Theorem 6 A unification problem $G =? H$ of type 0 has a solution if and only if the linear Diophantine equations in $Eq(G =? H)$ have a solution in non-negative integers.

Thus, the general unification problem of type 0 reduces to the problem of solving homogeneous and inhomogeneous linear Diophantine equations in non-negative integers. Efficient algorithms for the latter have recently been given by Clausen and Fortenbacher [4] and by Boudet, Contejean, and Devie [3, 5]. This problem is equivalent to the unification problem for expressions built up from an associative-commutative function symbol, a unit element, and free constants (see [14]).

Proof of Theorem 6. “Only if”: Let $\sigma = \{x_1/F_1, \dots, x_k/F_k\}$ be a solution to $G =? H$. Without loss of generality, we may assume that F_1, \dots, F_k are variable-free (otherwise $\sigma\tau$ is as required, where τ removes all variable hyperedges). Let $C_1, \dots, C_m, C_{m+1}, \dots, C_n$ be the non-isomorphic, connected, variable-free components occurring in F_1, \dots, F_k, G , and H . Then, for $i = 1, \dots, k$, F_i can be represented as the disjoint union of its connected components: $F_i = \sum_{j=1}^n c_{i,j} C_j$,

where the $c_{i,j}$ are non-negative integers. By setting $b_j = 0$ for $j = m + 1, \dots, n$ we get the following:

$$G\sigma = \sum_{i=1}^k a_i \left(\sum_{j=1}^n c_{i,j} C_j \right) + \sum_{j=1}^n b_j C_j = \sum_{j=1}^n \sum_{i=1}^k (a_i c_{i,j} + b_j) C_j$$

$$H\sigma = \sum_{i=1}^k a'_i \left(\sum_{j=1}^n c_{i,j} C_j \right) + \sum_{j=1}^n b'_j C_j = \sum_{j=1}^n \sum_{i=1}^k (a'_i c_{i,j} + b'_j) C_j$$

Since C_1, \dots, C_n are non-isomorphic, connected components, $G\sigma \cong H\sigma$ implies $\sum_{i=1}^k (a_i c_{i,j} + b_j) = \sum_{i=1}^k (a'_i c_{i,j} + b'_j)$ for $j = 1, \dots, n$. Thus, for $j = 1, \dots, m$, $(c_{1,j}, \dots, c_{k,j})$ is a solution of $\sum_{i=1}^k (a_i - a'_i) x_{i,j} = (b'_j - b_j)$ in non-negative integers.

“If”: Let, for $j = 1, \dots, m$, $(c_{1,j}, \dots, c_{k,j})$ be a solution of $\sum_{i=1}^k (a_i - a'_i) x_{i,j} = (b'_j - b_j)$ in non-negative integers. Consider the substitution $\sigma = \{x_i / \sum_{j=1}^m c_{i,j} C_j \mid i = 1, \dots, k\}$. Then:

$$G\sigma = \sum_{i=1}^k a_i \left(\sum_{j=1}^m c_{i,j} C_j \right) + \sum_{j=1}^m b_j C_j = \sum_{j=1}^m \left(\sum_{i=1}^k a_i c_{i,j} + b_j \right) C_j$$

$$H\sigma = \sum_{i=1}^k a'_i \left(\sum_{j=1}^m c_{i,j} C_j \right) + \sum_{j=1}^m b'_j C_j = \sum_{j=1}^m \left(\sum_{i=1}^k a'_i c_{i,j} + b'_j \right) C_j$$

Since, for $j = 1, \dots, m$, $(c_{1,j}, \dots, c_{k,j})$ is a solution of $\sum_{i=1}^k (a_i - a'_i) x_{i,j} = (b'_j - b_j)$, we get $\sum_{i=1}^k a_i c_{i,j} + b_j = \sum_{i=1}^k a'_i c_{i,j} + b'_j$. It follows $G\sigma \cong H\sigma$, that is, σ is a solution to $G \stackrel{?}{=} H$. \square

We now come back to unification problems of arbitrary type. In order to give sufficient conditions for the existence of solutions, we consider substitutions that reduce the type of unification problems.

Call a non-empty substitution $\sigma = \{x_1/G_1, \dots, x_n/G_n\}$ *simplifying* if, for $i = 1, \dots, n$, G_i is a hypergraph with *type*(x_i) nodes and either no hyperedge or one variable hyperedge of type less than *type*(x_i). Given a unification problem $G \stackrel{?}{=} H$, we denote by $Simplify(G \stackrel{?}{=} H)$ the set of all unification problems $G\rho \stackrel{?}{=} H\rho$ such that ρ is simplifying.

Proposition 7 *A unification problem $G \stackrel{?}{=} H$ has a solution if some problem in $Simplify(G \stackrel{?}{=} H)$ has a solution.*

Proof. Let σ be a solution to a problem $G\rho \stackrel{?}{=} H\rho$ in $Simplify(G \stackrel{?}{=} H)$, where ρ is a simplifying substitution. Then, by Lemma 2, $G(\rho\sigma) \cong (G\rho)\sigma \cong (H\rho)\sigma \cong H(\rho\sigma)$ and hence $\rho\sigma$ is a solution to $G \stackrel{?}{=} H$. \square

There are two particularly simple cases in which Proposition 7 guarantees that a problem $G =^? H$ has a solution: (1) $G^\ominus =^? H^\ominus$ has a solution (i.e. $G^\ominus \cong H^\ominus$) and (2) $G^0 =^? H^0$ has a solution, where G^0 and H^0 are obtained from G and H by substituting type-0 variables for all variables. The following proposition, which gives a syntactic condition for the existence of a solution, can also be seen as a consequence of Proposition 7.

Proposition 8 *A unification problem $G =^? H$ has a solution if there are variables $x \in \text{Var}(G) - \text{Var}(H)$ and $y \in \text{Var}(H) - \text{Var}(G)$ such that x and y occur only once in G and H , respectively.*

Proof. Define a substitution $\sigma = \{z/\sigma(z) \mid z \in \text{Var}(G) \cup \text{Var}(H)\}$ by

$$\sigma(z) = \begin{cases} \text{type}(x)^\bullet + H^\ominus & \text{if } z = x, \\ \text{type}(y)^\bullet + G^\ominus & \text{if } z = y, \\ \text{type}(z)^\bullet & \text{otherwise,} \end{cases}$$

where for each $n \geq 0$, n^\bullet is a hypergraph with n nodes and no hyperedges. Then σ is a solution to $G =^? H$ since $G\sigma \cong G^\ominus + H^\ominus \cong H\sigma$. \square

The converse of Proposition 7 does not hold. In fact, there are unification problems of arbitrary type that cannot be solved via simplification.

Proposition 9 *For each $n \geq 1$, there is a unification problem $G =^? H$ of type n that has a solution while no problem in $\text{Simplify}(G =^? H)$ has a solution.*

Proof. The unification problem $G =^? H$ given in Figure 3 obviously has a solution. But each problem in $\text{Simplify}(G =^? H)$ is of the form $G' =^? H$, where G' contains at least one isolated node (a node to which no hyperedge is attached). Since H does not contain isolated nodes and is variable-free, there is no solution to $G' =^? H$. \square

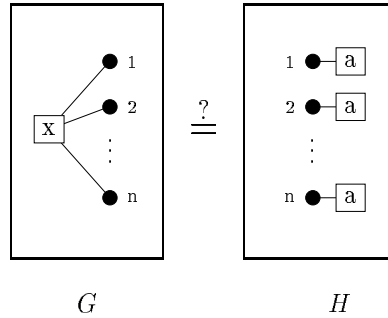


Figure 3: Unification problem for the proof of Proposition 9

4 Matching

We now turn from unification to matching (with variables of arbitrary type).

Definition 10 A *matching problem* $G \leq^? H$ consists of two hypergraphs G and H such that H is variable-free. A *solution* to this problem is a substitution σ such that $G\sigma \cong H$. The solution is *pure* if $Dom(\sigma) = Var(G)$.

The assumption that H is variable-free can be made without loss of generality, for if it is violated one considers a problem $G \leq^? H'$ in which H' is obtained from H by renaming each variable into a new non-variable label. Then the solutions to $G \leq^? H$ can be obtained by applying the inverse transformation to the hypergraphs in the solutions to $G \leq^? H'$.

In the following we do not distinguish between substitutions that are equal up to isomorphism. More precisely, two substitution pairs x/G and x/H are considered to be equal if there is an isomorphism $f: G \rightarrow H$ with $f^*(p_G) = p_H$. (Recall that p_G and P_G denote the sequence and the set of points, respectively, of a pointed hypergraph G .)

Definition 11 Let $G \leq^? H$ be a matching problem and e be a hyperedge in G with $lab_G(e) = x \in Var(G)$. Then a pointed hypergraph C with $type(C) = type(x)$ is a *candidate* for e if there is an injective hypergraph morphism $in: G^\ominus \rightarrow H$ and a hypergraph morphism $f: C \rightarrow H$ such that the following conditions are satisfied:

- (1) $in^*(att_G(e)) = f^*(p_C)$,
- (2) the subhypergraphs $in(G^\ominus)$ and $f(C)$ of H overlap only in $f(P_C)$, that is, $in(G^\ominus) \cap f(C) = f(P_C)$,
- (3) for all items c, c' in C with $c \neq c'$, $f(c) = f(c')$ implies $c, c' \in P_C$,
- (4) no hyperedge in $E_H - f(E_C)$ is incident to a node in $f(V_C - P_C)$.

Let **MATCH** be the set of transformation rules shown in Figure 4. These rules

TRY	$\langle G \leq^? H, \sigma \rangle \Rightarrow \langle G\{x/C\} \leq^? H, \sigma\{x/C\} \rangle$	if $x \in Var(G)$ and C is a candidate for some x -labelled hyperedge in G
FAIL 1	$\langle G \leq^? H, \sigma \rangle \Rightarrow \mathbf{F}$	if there is no injective morphism $G^\ominus \rightarrow H$
FAIL 2	$\langle G \leq^? H, \sigma \rangle \Rightarrow \mathbf{F}$	if $Var(G) = \emptyset$ and $G \not\cong H$

Figure 4: The rule set **MATCH**

operate on pairs of matching problems and substitutions. The transformation

relation defined by repeated application of `MATCH` rules is denoted by $\Rightarrow^*_{\text{MATCH}}$. By the following result, all pure solutions of a problem $G \leq^? H$ can be obtained by computing all `MATCH` derivations starting from the pair $\langle G \leq^? H, \emptyset \rangle$.

Theorem 12 *A substitution σ is a pure solution to a matching problem $G \leq^? H$ if, and only if, $\langle G \leq^? H, \emptyset \rangle \Rightarrow^*_{\text{MATCH}} \langle M \leq^? H, \sigma \rangle$ such that no `MATCH` rule is applicable to $\langle M \leq^? H, \sigma \rangle$.*

The proof of this result is given in two parts which establish soundness and completeness of the `MATCH` rules.

Lemma 13 (soundness) *If $\langle G \leq^? H, \emptyset \rangle \Rightarrow^*_{\text{MATCH}} \langle M \leq^? H, \sigma \rangle$ such that no `MATCH` rule is applicable to $\langle M \leq^? H, \sigma \rangle$, then σ is a pure solution to $G \leq^? H$.*

Proof. Since `FAIL 1` is not applicable, there is an injective morphism $M^\ominus \rightarrow H$. Then $\text{Var}(M) = \emptyset$ as otherwise the attachment nodes of a variable hyperedge would form a candidate for this hyperedge, implying that `TRY` were applicable. It follows that M and H are isomorphic, because `FAIL 2` is not applicable.

Now consider an arbitrary derivation $\langle A \leq^? B, \emptyset \rangle \Rightarrow^*_{\text{TRY}} \langle A' \leq^? B, \alpha \rangle$. By using the definition of `TRY`, an easy induction on the length of this derivation shows that $A\alpha \cong A'$ and $\text{Dom}(\alpha) \subseteq \text{Var}(A)$ hold. Thus $G\sigma \cong M \cong H$ and $\text{Dom}(\sigma) = \text{Var}(G)$, that is, σ is a pure solution to $G \leq^? H$. \square

Lemma 14 (completeness) *If σ is a pure solution to a matching problem $G \leq^? H$, then $\langle G \leq^? H, \emptyset \rangle \Rightarrow^*_{\text{TRY}} \langle G\sigma \leq^? H, \sigma \rangle$.*

Proof. We proceed by induction on the size of $\text{Var}(G)$. If $\text{Var}(G) = \emptyset$, then $G \cong H$ and the `TRY` derivation of length 0 is as required. Now let $\text{Var}(G) \neq \emptyset$ and suppose, as induction hypothesis, that the proposition holds for every matching problem $G' \leq^? H$ with $|\text{Var}(G')| < |\text{Var}(G)|$. Consider some substitution pair x/C in σ and let $\tau = \sigma - \{x/C\}$. By Lemma 2 and the fact that C is variable-free,

$$\begin{aligned} (G\{x/C\})\tau &= G(\{x/C\}\tau) \\ &= G(\{x/C\} \cup \tau) \\ &= G\sigma \\ &\cong H. \end{aligned}$$

So τ is a pure solution to $G\{x/C\} \leq^? H$. Hence, by induction hypothesis,

$$\langle G\{x/C\} \leq^? H, \emptyset \rangle \Rightarrow^*_{\text{TRY}} \langle (G\{x/C\})\tau \leq^? H, \tau \rangle.$$

Moreover, a straightforward induction on the length of derivations shows that every derivation $\langle A \leq^? B, \emptyset \rangle \Rightarrow^*_{\text{TRY}} \langle A' \leq^? B, \alpha \rangle$ can be transformed into a derivation $\langle A \leq^? B, \beta \rangle \Rightarrow^*_{\text{TRY}} \langle A' \leq^? B, \beta\alpha \rangle$, where β is an arbitrary substitution. Thus

$$\langle G\{x/C\} \leq^? H, \{x/C\} \rangle \Rightarrow^*_{\text{TRY}} \langle (G\{x/C\})\tau \leq^? H, \{x/C\}\tau \rangle.$$

It is easy to check that C is a candidate for each x -labelled hyperedge in G , implying $\langle G \leq^? H, \emptyset \rangle \Rightarrow_{\text{TRY}} \langle G\{x/C\} \leq^? H, \{x/C\} \rangle$. Therefore, by transitivity of $\Rightarrow_{\text{TRY}}^*$

$$\langle G \leq^? H, \emptyset \rangle \Rightarrow_{\text{TRY}}^* \langle (G\{x/C\})\tau \leq^? H, \{x/C\}\tau \rangle.$$

This completes the proof since $(G\{x/C\})\tau \cong G\sigma$ and $\{x/C\}\tau = \sigma$. \square

Corollary 15 *There is an algorithm that computes for every matching problem all pure solutions.*

Proof. Every sequence of **MATCH** applications terminates since for every step $\langle G \leq^? H, \sigma \rangle \Rightarrow_{\text{TRY}} \langle G' \leq^? H, \sigma' \rangle$, G' contains fewer variables than G . Moreover, from every pair $\langle G \leq^? H, \sigma \rangle$ only finitely many pairs can be generated by a **TRY** step. For there are, up to isomorphism, only finitely many candidates for each variable hyperedge in G . It follows that the set of **MATCH** derivations starting from a pair $\langle G \leq^? H, \emptyset \rangle$ is finite, and hence it can be computed. Call a derivation in this set successful if it ends with a pair to which no **MATCH** rule can be applied. By Theorem 12, the substitutions in the final pairs of the successful derivations are the pure solutions of $G \leq^? H$. \square

5 Substitution-based graph rewriting

The above considerations suggest a hypergraph rewriting mechanism based on matching. The approach is very simple to describe and needs neither pushouts nor embedding instructions: A *rule* is just a pair $\langle L, R \rangle$ of hypergraphs which is applied to a hypergraph G by (1) choosing a solution σ to the matching problem $L \leq^? G$ and (2) applying σ to R .

Example 16 Figure 5 shows a rule for program transformation on flow graphs,

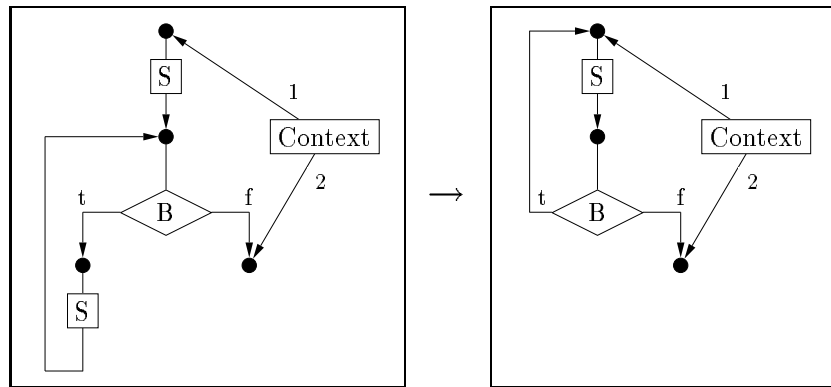


Figure 5: A rule

where B , S , and Context are variables. The rule can transform a pattern of the form “ S ; while B do S ” into “repeat S until $\neg B$ ”.

In order to state that substitution-based rewriting generalizes the well-known double-pushout approach [7], recall that a *DPO-rule* $r = \langle L \leftarrow K \rightarrow R \rangle$ consists of two hypergraph morphisms with a common domain. The application of r to a hypergraph G consists of the construction of the two hypergraph pushouts shown in Figure 6. Without loss of generality we may assume that the vertical

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

Figure 6: A double-pushout

morphisms in these pushouts are injective, since r can be translated into a finite set $S(r)$ of DPO-rules such that each application of r corresponds to an application of a rule in $S(r)$ obeying the injectivity condition and vice versa.

Given a DPO-rule $r = \langle L \leftarrow K \rightarrow R \rangle$, we construct a rule $r^\oplus = \langle L^\oplus, R^\oplus \rangle$ as follows: Let K^\oplus be the extension of K by a variable-hyperedge of type $|V_K|$ which is attached to all nodes of K . Then L^\oplus is the pushout object of L and K^\oplus along K , and R^\oplus is the pushout object of R and K^\oplus along K . When r^\oplus is applied to a hypergraph G , the variable in L^\oplus is replaced by the context of L in G .

Theorem 17 *Let G and H be hypergraphs and r be a DPO-rule. Then H results from an application of r to G in the double-pushout approach if, and only if, H results from an application of r^\oplus to G in the substitution-based approach.*

So the double-pushout approach is included in the substitution-based approach. But the latter is a proper generalization of the former as there are rules which remove or copy non-local parts of the hypergraphs they are applied to. This is simply achieved by omitting or duplicating variables from the left-hand side of a rule in the right-hand side. For instance, the rule in Example 16 cannot be simulated by a double-pushout rule. This is because there are applications of the rule with arbitrarily large subhypergraphs matched by the S -variables. So the rule can remove a subhypergraph of unbounded size whereas the number of items removed by a double-pushout rule is bounded by the size of its left-hand side.

Proof of Theorem 17. Let H result from an application of r to G in the double-pushout approach as shown in Figure 6. Let K_0 be the discrete hypergraph obtained from K by removing all hyperedges, and D_0 be the hypergraph obtained from D by removing the images of all hyperedges of K . Then there are inclusion

morphisms $K_0 \rightarrow K$ and $D_0 \rightarrow D$, and $K \rightarrow D$ can be restricted to $K_0 \rightarrow D_0$. By assumption, the vertical morphisms in Figure 6 are injective. So D is the pushout object of K and D_0 along K_0 . Correspondingly, G and H are the pushout objects of L and D_0 along K_0 , and of R and D_0 along K_0 , respectively.

Now let $\sigma = \{x/\langle D_0, p_0 \rangle\}$, where x is the label of the variable hyperedge in K_0^\oplus , and p_0 is the image of the attachment sequence of this hyperedge. Then, obviously, $K_0^\oplus \sigma \cong D_0$. The following relation between pushouts and substitutions is easy to show: if D is the pushout object of B and C along A , and $A^\oplus \sigma \cong B$, then $C^\oplus \sigma \cong D$. Hence we obtain $L^\oplus \sigma \cong G$ and $R^\oplus \sigma \cong H$, that is, H results from an application of r^\oplus to G in the substitution-based approach.

Conversely, let H result from an application of r^\oplus to G in the substitution-based approach. Then $L^\oplus \sigma \cong G$ and $R^\oplus \sigma \cong H$ for some substitution σ . Let $D = K^\oplus \sigma$, G' be the pushout object of L and D along K , and H' be the pushout object of R and D along K . By the above mentioned relation between pushouts and substitutions, we have $G' \cong L^\oplus \sigma \cong G$ and $H' \cong R^\oplus \sigma \cong H$. Thus, H results from an application of r to G in the double-pushout approach. \square

6 Conclusion

The decidability of the general graph unification problem is an intriguing theoretical problem. It is easy to state but turns out to be non-trivial already for the very restricted case of type 0 variables. The general case may be hard to solve in view of Makanin's extremely complex proof of the decidability of word unification [16] (see also [13, 19]).

On the other hand, it is not clear whether word unification can be simulated by graph unification. The problem is that for the obvious graph representation of words as chains of edges, unifying graph substitutions need not preserve this structure and hence may not correspond to word substitutions. For example, the word unification problem $\mathbf{ax} =? \mathbf{by}$ would have a graph solution (see Proposition 8) although the two words are not unifiable. For similar reasons it is unclear whether graph unification can simulate second-order term unification. Since the latter is undecidable [8], this simulation would show that graph unification is undecidable.

Instead of representing words or terms as graphs, one can also go the reverse direction. Bauderon and Courcelle [2] define three graph operations by which hypergraphs can be represented as first-order terms over a given set of typed labels. They give eleven equations which equate each two different terms that represent the same hypergraph. By providing such terms with variables for hyperedges, graph unification can be rephrased as term unification modulo the equational theory induced by the eleven equations.

References

- [1] Franz Baader and Jörg H. Siekmann. Unification theory. In D. M. Gabbay,

- C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994.
- [2] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20:83–127, 1987.
 - [3] Alexandre Boudet, Evelyne Contejean, and Hervé Devie. A new AC unification algorithm with an algorithm for solving systems of Diophantine equations. In *Proc. Annual IEEE Symposium on Logic in Computer Science*, pages 289–299, 1990.
 - [4] Michael Clausen and Albrecht Fortenbacher. Efficient solution of linear Diophantine equations. *Journal of Symbolic Computation*, 8:201–216, 1989.
 - [5] Evelyne Contejean. An efficient incremental algorithm for solving systems of linear Diophantine equations. *Information and Computation*, 113:143–172, 1994.
 - [6] Jacques Corbin and Michel Bidoit. A rehabilitation of Robinson’s unification algorithm. In R.E.A. Mason, editor, *Proc. Information Processing 83*, pages 909–914, 1983.
 - [7] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In *Proc. Graph-Grammars and Their Application to Computer Science and Biology*, pages 1–69. Springer Lecture Notes in Computer Science 73, 1979.
 - [8] Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
 - [9] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
 - [10] Annegret Habel and Detlef Plump. Unification, rewriting, and narrowing on term graphs. In *Proc. SEGRAGRA ’95*, Electronic Notes in Theoretical Computer Science. Elsevier, 1995. To appear.
 - [11] Gérard P. Huet. Résolution d’équations dans les langages d’ordre $1, 2, \dots, \omega$. Thèse d’état, Université de Paris VII, 1976.
 - [12] Joxan Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2:207–219, 1984.
 - [13] Joxan Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, 1990.
 - [14] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

- [15] Kevin Knight. Unification: A multidisciplinary survey. *ACM Computing Surveys*, 21(1):93–124, 1989.
- [16] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 103:147–236, 1977. In Russian. English translation in *Math. USSR Sbornik*, 32:129–198, 1977.
- [17] Francesco Parisi-Presicce, Hartmut Ehrig, and Ugo Montanari. Graph rewriting with unification and composition. In *Proc. Graph-Grammars and Their Application to Computer Science*, pages 496–514. Springer Lecture Notes in Computer Science 291, 1987.
- [18] M.S. Paterson and M.N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16:158–167, 1978.
- [19] Klaus U. Schulz. Word unification and transformation of generalized equations. *Journal of Automated Reasoning*, 11:149–184, 1993.