On the Parameterized Complexity of Associative and Commutative Unification

Tatsuya Akutsu[1(✉)], Jesper Jansson[1,2], Atsuhiro Takasu[3], and Takeyuki Tamura[1]

[1] Bioinformatics Center, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan
{takutsu,jj,tamura}@kuicr.kyoto-u.ac.jp
[2] The Hakubi Project, Kyoto University, Sakyo-ku, Kyoto 606-8501, Japan
[3] National Institute of Informatics, Tokyo 101-8430, Japan
takasu@nii.ac.jp

Abstract. This paper studies the unification problem with associative, commutative, and associative-commutative functions. The parameterized complexity is analyzed with respect to the parameter "number of variables". It is shown that both the associative and associative-commutative unification problems are $W[1]$-hard. For commutative unification, a polynomial-time algorithm is presented in which the number of variables is assumed to be a constant. Some related results for the string and tree edit distance problems with variables are also presented.

1 Introduction

Unification is an important concept in many areas of computer science such as automated theorem proving, program verification, natural language processing, logic programming, and database query systems [14,17,18]. The unification problem is, in its fundamental form, to find a substitution for all variables in two given terms that make the terms identical, where terms are built up from function symbols, variables, and constants [18]. As an example, the two terms $f(x, y)$ and $f(g(a), f(b, x))$ with variables x and y and constants a and b become identical by substituting x by $g(a)$ and y by $f(b, g(a))$. When one of the two input terms contains no variables, the unification problem is called *matching*.

Unification has a long history beginning with the seminal work of Herbrand in 1930 (see, e.g., [18]). It is becoming an active research area again because of *math search*, an information retrieval (IR) task where the objective is to find all documents containing a specified mathematical formula and/or all formulas similar to a query formula [16,19,20]. Also, math search systems such as Wolfram Formula Search and Wikipedia Formula Search have been developed. Since mathematical formulas are typically represented by rooted trees, it seems natural to measure the similarity between formulas by measuring the structural

This work was partially supported by the Collaborative Research Programs of National Institute of Informatics.

M. Cygan and P. Heggernes (Eds.): IPEC 2014, LNCS 8894, pp. 15–27, 2014.
DOI: 10.1007/978-3-319-13524-3_2

similarity of their trees. However, methods based on approximate tree matching like the *tree edit distance* (see, e.g., the survey in [7]) alone are not sufficient since every label is treated as a constant. For example, the query $x^2 + x$ has the same tree edit distance to each of the formulas $y^2 + z$ and $y^2 + y$ although $y^2 + y$ is mathematically the same as $x^2 + x$, but $y^2 + z$ is not.

An exponential-time algorithm for the unification problem was given in [22] and a faster, linear-time algorithm [9,21] appeared a few years later. Various extensions of unification have also been considered in the literature [5,17,18]. Three of them, unification with commutative, associative, and associative-commutative functions (where a function f is called *commutative* if $f(x, y) = f(y, x)$ always holds, *associative* if $f(x, f(y, z)) = f(f(x, y), z)$ always holds, and *associative-commutative* if it is both associative and commutative), are especially relevant for math search since many functions encountered in practice have one of these properties. However, when allowing such functions, there are more ways to match nodes in the two corresponding trees, and as a result, the computational complexity of unification may increase. Indeed, each of the associative, commutative, and associative-commutative unification (and matching) problems is NP-hard [5,10,17], and polynomial-time algorithms are known only for very restricted cases [2,5,17]; e.g., associative-commutative matching can be done in polynomial time if every variable occurs exactly once [5]. Due to the practical importance of these (and other) extensions of unification, heuristic algorithms have been proposed, sometimes incorporating approximate tree matching techniques [13,14].

This paper studies the parameterized complexity of associative, commutative, and associative-commutative unification with respect to the parameter "number of variables appearing in the input", denoted from here on by k. (We choose this parameter because the number of variables is often much smaller than the size of the terms.) In addition, we introduce and study the string and tree edit distance problems with variables. The following table summarizes our new results:

	Matching	Unification	DO-matching	DO-unification
SEDV	$W[2]$-hard (Theorem 1)	$O(\|\Sigma\|^k poly)$ (Proposition 1)	–	P (Theorem 4)
OTEDV	$W[1]$-hard (Theorem 3)	–	–	P (Theorem 4)
Associative	$W[1]$-hard (Theorem 5) (NP-complete [5])	–	P [5]	P (Theorem 6)
Commutative	NP-hard [5] FPT[a] (Theorem 7)	XP (Theorem 8)	P [5]	P (Proposition 3)
Associative and commutative	$W[1]$-hard (Theorem 9) (NP-hard [5])	–	P [5]	P (Proposition 4)

[a] Under the assumption that Conjecture 1 holds

Here, SEDV = the string edit distance problem with variables, OTEDV = the ordered tree edit distance problem with variables, and DO = distinct occurrences of all variables. $W[1]$-hard and FPT mean with respect to the parameter k. For simplicity, the algorithms described in this paper only determine if two terms are unifiable, but they may be modified to output the corresponding substitutions (when unifiable) by using standard traceback techniques. We remark that associative unification is in PSPACE and both commutative unification and associative-commutative unification are in NP [6]; although it means that all problems can be solved in single exponential time of the size of the input, it does not necessarily mean single exponential-time algorithms with respect to the number of variables.

2 Unification of Strings

Let Σ be an alphabet and Γ a set of variables. A *substitution* is a mapping from Γ to Σ. For any string s over $\Sigma \cup \Gamma$ and substitution θ, let $s\theta$ denote the string over Σ obtained by replacing every occurrence of a variable $x \in \Gamma$ in s by the symbol $\theta(x)$. (We write x/a to express that x is substituted by a.) Two strings s_1 and s_2 are called *unifiable* if there exists a substitution θ such that $s_1\theta = s_2\theta$.

Example 1. Suppose $\Sigma = \{a, b, c\}$ and $\Gamma = \{x, y, z\}$. Let $s_1 = abxbx$, $s_2 = ayczc$, and $s_3 = ayczb$. Then s_1 and s_2 are unifiable since $s_1\theta = s_2\theta = abcbc$ holds for $\theta = \{x/c,\ y/b,\ z/b\}$. On the other hand, s_1 and s_3 are not unifiable since there does not exist any θ with $s_1\theta = s_3\theta$. □

We shall use the following notation. For any string s, $|s|$ is the length of s. For any two strings s and t, the string obtained by concatenating s and t is written as $s\,t$. Furthermore, for any positive integers i, j with $1 \leq i \leq j \leq |s|$, $s[i]$ is the ith character of s and $s[i \ldots j]$ is the substring $s[i]\,s[i+1] \cdots s[j]$. (Thus, $s = s[1..|s|]$.) The *string edit distance* (see, e.g., [15]) between two strings s_1, s_2 over Σ, denoted by $d_S(s_1, s_2)$, is the length of a shortest sequence of edit operations that transforms s_1 into s_2, where an *edit operation* on a string is one of the following three operations: a *deletion* of the character at some specified position, an *insertion* of a character at some specified position, or a *replacement* of the character at some specified position by a specified character.[1] For example, $d_S(bcdfe, abgde) = 3$ because $abgde$ can be obtained from $bcdfe$ by the deletion of f, the replacement of c by g, and the insertion of an a, and no shorter sequence can accomplish this. By definition, $d_S(s_1, s_2) = \min_{ed\,:\,ed(s_1)=s_2} |ed| = \min_{ed\,:\,ed(s_2)=s_1} |ed|$ holds, where ed is a sequence of edit operations.

We generalize the string edit distance to two strings s_1, s_2 over $\Sigma \cup \Gamma$ by defining $\hat{d}_S(s_1, s_2) = \min_{ed\,:\,(\exists\theta)\,(ed(s_1)\theta = s_2\theta)} |ed|$. The *string edit distance problem with variables* takes as input two strings s_1, s_2 over $\Sigma \cup \Gamma$, and asks for the value of $\hat{d}_S(s_1, s_2)$. (To the authors' knowledge, this problem has not been

[1] In the literature, "replacement" is usually referred to as "substitution". Here, we use "replacement" to distinguish it from the "substitution" of variables defined above.

studied before. Note that it differs from the *pattern matching with variables problem* [11], in which one of the two input strings contains no variables and each variable may be substituted by any string over Σ, but no insertions or deletions are allowed.) Let k be the number of variables appearing in at least one of s_1 and s_2. Although $d_S(s_1, s_2)$ is easy to compute in polynomial time (see [15]), computing $\hat{d}_S(s_1, s_2)$ is $W[2]$-hard with respect to the parameter k:

Theorem 1. *The string edit distance problem with variables is $W[2]$-hard with respect to k when the number of occurrences of every variable is unrestricted.*

Proof. We present an FPT-reduction [12] from the longest common subsequence problem (LCS) to a decision problem version of the edit distance problem with variables. LCS is, given a set of strings $R = \{r_1, r_2, \ldots, r_q\}$ over an alphabet Σ_0 and an integer l, to determine whether there exists a string r of length l such that r is a subsequence of r_i for every $r_i \in R$, where r is called a *subsequence* of r' if r can be obtained by performing deletion operations on r'. It is known that LCS is $W[2]$-hard with respect to the parameter l (problem "LCS-2" in [8]).

Given any instance of LCS, we construct an instance of the string edit distance problem with variables as follows. Let $\Sigma = \Sigma_0 \cup \{\#\}$, where $\#$ is a symbol not appearing in $r_1, r_2, \ldots, r_q$, and $\Gamma = \{x_1, x_2, \ldots, x_l\}$. Clearly, R has a common subsequence of length l if and only if there exists a θ such that $x_1 x_2 \cdots x_l \theta$ is a common subsequence of R. Now, construct s_1 and s_2 by setting:

$$s_1 = x_1 x_2 \cdots x_l \# x_1 x_2 \cdots x_l \# \cdots \# x_1 x_2 \cdots x_l$$
$$s_2 = r_1 \# r_2 \# \cdots \# r_q$$

where the substring $x_1 x_2 \cdots x_l$ occurs q times in s_1. By the construction, there exists a θ such that $x_1 x_2 \cdots x_l \theta$ is a common subsequence of R if and only if there exists a θ such that $s_1\theta$ is a subsequence of s_2. The latter statement holds if and only if $\hat{d}_S(s_1, s_2) = (\sum_{i=1}^{q} |r_i|) - ql$. Since $k = l$, this is an FPT-reduction. □

The above proof can be extended to prove the $W[1]$-hardness of a restricted case with a bounded number of occurrences of each variable (omitted in the conference proceedings version).

Theorem 2. *The string edit distance problem with variables is $W[1]$-hard with respect to k, even if the total number of occurrences of every variable is* 2.

Note that in the special case where every variable in the input occurs exactly once, the problem is equivalent to approximate string matching with don't-care symbols, which can be solved in polynomial time [3].

On the positive side, the number of possible θ is bounded by $|\Sigma|^k$. This immediately yields a fixed-parameter algorithm w.r.t. k when Σ is fixed:

Proposition 1. *The string edit distance problem with variables can be solved in $O(|\Sigma|^k poly(m, n))$ time, where m and n are the lengths of the two input strings.*

3 Unification of Terms

We now consider the concept of unification for structures known as *terms* that are more general than strings [18]. From here on, Σ is a set of function symbols, where each function symbol has an associated *arity*, which is an integer describing how many arguments the function takes. A function symbol with arity 0 is called a *constant*. Γ is a set of variables. A *term* over $\Sigma \cup \Gamma$ is defined recursively as: (i) A constant is a term; (ii) A variable is a term; (iii) If $t_1, \ldots, t_d$ are terms and f is a function symbol with arity $d > 0$ then $f(t_1, \ldots, t_d)$ is a term.

Every term is identified with a rooted, ordered, node-labeled tree in which every internal node corresponds to a function symbol and every leaf corresponds to a constant or a variable. The tree identified with a term t is also denoted by t. For any term t, $N(t)$ is the set of all nodes in its tree t, $r(t)$ is the root of t, and $\gamma(t)$ is the function symbol of $r(t)$. The *size* of t is defined as $|N(t)|$. For any $u \in N(t)$, t_u denotes the subtree of t rooted at u and hence corresponds to a subterm of t. Any variable that occurs only once in a term is called a *DO-variable*, where "DO" stands for "distinct occurrences", and a term in which all variables are DO-variables is called a *DO-term* [5]. A term that consists entirely of elements from Σ is called *variable-free*.

Let $\mathcal{T}$ be a set of terms over $\Sigma \cup \Gamma$. A *substitution* θ is defined as any partial mapping from Γ to $\mathcal{T}$ (where we let x/t indicate that the variable x is mapped to the term t), under the constraint that if $x/t \in \theta$ then t is not allowed to contain the variable x. For any term $t \in \mathcal{T}$ and substitution θ, $t\theta$ is the term obtained by simultaneously replacing its variables in accordance with θ. For example, $\theta = \{x/y,\ y/x\}$ is a valid substitution, and in this case, $f(x, y)\theta = f(y, x)$.

Two terms $t_1, t_2 \in \mathcal{T}$ are said to be *unifiable* if there exists a θ such that $t_1\theta = t_2\theta$, and such a θ is called a *unifier*. In this paper, the *unification problem* is to determine whether two input terms t_1 and t_2 are unifiable. (Other versions of the unification problem have also been studied in the literature, but will not be considered here.) Unless otherwise stated, m and n denote the sizes of the two input terms t_1 and t_2. The unification problem can be solved in linear time [9,21]. The important special case of the unification problem where one of the two input terms is variable-free is called the *matching problem*.

Example 2. Let $\Sigma = \{a, b, f, g\}$, where a and b are constants, f has arity 2, and g has arity 3, and let $\Gamma = \{w, x, y, z\}$. Define the terms $t_1 = f(g(a, b, a), f(x, x))$, $t_2 = f(g(y, b, y), z)$, and $t_3 = f(g(a, b, a), f(w, f(w, w)))$. Then t_1 and t_2 are unifiable since $t_1\theta_1 = t_2\theta_1 = f(g(a, b, a), f(x, x))$ holds for $\theta_1 = \{y/a,\ z/f(x, x)\}$. Similarly, t_2 and t_3 are unifiable since $t_2\theta_2 = t_3\theta_2 = f(g(a, b, a), f(w, f(w, w)))$ with $\theta_2 = \{y/a,\ z/f(w, f(w, w))\}$. However, t_1 and t_3 are not unifiable because it is impossible to simultaneously satisfy $x = w$ and $x = f(w, w)$. □

Similar to what was done in Sect. 2, we can combine the *tree edit distance* with unification to get what we call the *tree edit distance problem with variables*. Let $d_T(t_1, t_2)$ be the tree edit distance between two node-labeled (ordered or unordered) trees t_1 and t_2 (see [7] for the definition). We generalize

$d_T(t_1, t_2)$ to two trees, i.e., two terms, over $\Sigma \cup \Gamma$ by defining $\hat{d}_T(t_1, t_2) = \min_{ed:(\exists\theta)\,(ed(t_1)\theta = t_2\theta)} |ed|$. The *tree edit distance problem with variables* takes as input two (ordered or unordered) trees t_1, t_2 over $\Sigma \cup \Gamma$, and asks for the value of $\hat{d}_T(t_1, t_2)$.

As before, let k be the number of variables appearing in at least one of t_1 and t_2. By combining the proofs of Theorems 2 and 5 below, we obtain:

Theorem 3. *The tree edit distance problem with variables is $W[1]$-hard with respect to k, both for ordered and unordered trees, even if the number of occurrences of every variable is bounded by 2.*

As demonstrated in [5], certain matching problems are easy to solve for DO-terms. The next theorem, whose proof is omitted in this version, states that the ordered tree edit distance problem with variables also becomes polynomial-time solvable for DO-terms. (In contrast, the classic *unordered* tree edit distance problem is already NP-hard for variable-free terms; see, e.g., [7].)

Theorem 4. *The ordered tree edit distance problem with variables can be solved in polynomial time when t_1 and t_2 are DO-terms.*

4 Associative Unification

A function f with arity 2 is called *associative* if $f(x, f(y, z)) = f(f(x, y), z)$ always holds. Associative unification is a variant of unification in which functions may be associative. This section assumes that all functions are associative although all results are valid by appropriately modifying the details even if usual (non-associative) functions are included.

Associative matching was shown to be NP-hard in [5] by a simple reduction from 3SAT. However, the proof in [5] does not show the parameterized hardness.

Theorem 5. *Associative matching is $W[1]$-hard with respect to the number of variables even for a fixed Σ.*

Proof. As in the proof of Theorem 1, we reduce from LCS.

First consider the case of an unrestricted Σ. Let $(\{r_1, \ldots, r_q\}, l)$ be any given instance of LCS. For each $i = 1, \ldots, q$, create a term u^i as follows: $u^i = f(y_{i,1}, f(x_1, f(y_{i,2}, f(x_2, \cdots f(y_{i,l}, f(x_l, f(y_{i,l+1}, g(\#, \#))) \cdots))))$, where $\#$ is a character not appearing in $r_1, \ldots, r_q$. Create a term t_1 by replacing the last occurrence of $\#$ in each u^i by u^{i+1} for $i = 1, \ldots, q-1$, thus concatenating $u^1, \ldots, u^q$, as shown in Fig. 1. Next, transform each r_i into a string r'_i of length $1 + 2 \cdot |r_i|$ by inserting a special character & in front of each character in r_i, and appending & to the end of r_i, where each & is considered to be a distinct constant (i.e., & does not match any symbol but can match any variable). Represent each r'_i by a term t^i defined by: $t^i = f(r'_i[1], f(r'_i[2], f(r'_i[3], f(\cdots, f(r'_i[1 + 2 \cdot |r'_i|], g(\#, \#)) \cdots))))$. Finally, create a term t_2 by concatenating $t^1, \ldots, t^q$. (Again, see Fig. 1.) Now, t_1 and t_2 are unifiable if and only if there exists a

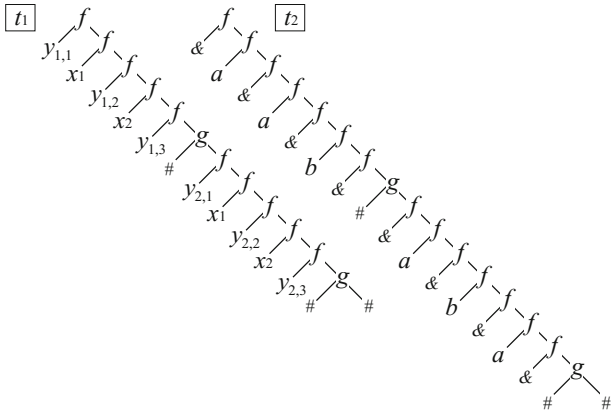

Fig. 1. Illustrating the reduction in Theorem 5. Here, $s_1 = aab$, $s_2 = aba$, and $l = 2$.

common subsequence of $\{r_1, \ldots, r_q\}$ of length l. Since the number of variables in t_1 is $(l+1)q + l = lq + l + q$, it is an FPT-reduction and thus the problem is $W[1]$-hard.

For the case of a fixed Σ, represent each constant by a distinct term using a special function symbol h and binary encoding (e.g., the 10th symbol among 16 symbols can be represented as $h(1, h(0, h(1, 0)))$). □

We next consider associative unification for DO-terms, which has some similarities with DO-associative-commutative matching [5]. For any term t, define the *canonical form* of t (called the "flattened form" in [5]) as the term obtained by contracting all edges in t whose two endpoints are labeled by the same function symbol. For example, both $f(f(a,b), f(g(c, f(d, f(e,h)), e))$ and $f(a, f(b, f(g(c, f(f(d,e),h)), e)))$ are transformed into $f(a, b, g(c, f(d, e, h)), e)$. As another example, the canonical form of $f(g(a,b), f(c,d))$ is $f(g(a,b), c, d)$. It is known [5] that the canonical form of t can be computed in linear time.

We begin with the simplest case in which every term is variable-free.

Proposition 2. *Associative unification for variable-free terms can be done in linear time.*

Proof. Transform the two terms into their canonical forms in linear time as above. Then it suffices to test if the canonical forms are isomorphic. The rooted ordered labeled tree isomorphism problem is trivially solvable in linear time. □

To handle the more general case of two DO-terms t_1 and t_2, we transform them into their canonical forms t^1 and t^2 and apply the following procedure, which returns 'true' if and only if t^1 and t^2 are unifiable. See Fig. 2 for an illustration. The procedure considers all $u \in N(t^1)$, $v \in N(t^2)$ in bottom-up order, and assigns $D[u, v] = 1$ if and only if $(t^1)_u$ and $(t^2)_v$ are unifiable.

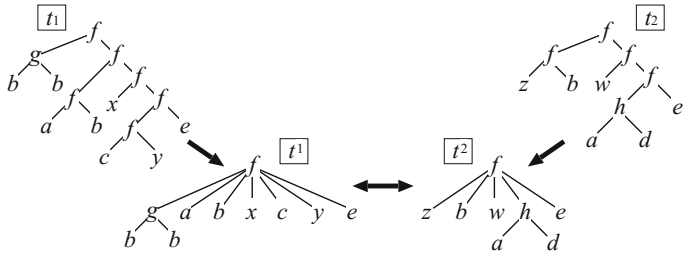

Fig. 2. An example of associative unification. The DO-terms t_1, t_2 are transformed into their canonical forms and then unified by $\theta = \{y/h(a,d),\ z/f(g(b,b),a),\ w/f(x,c)\}$.

Procedure $AssocMatchDO(t^1, t^2)$
  for all $u \in N(t^1)$ **do** /* in post-order */
    for all $v \in N(t^2)$ **do** /* in post-order */
      if $(t^1)_u$ or $(t^2)_v$ is a constant **then**
        if $(t^1)_u$ and $(t^2)_v$ are unifiable (#)
        then $D[u,v] \leftarrow 1$ **else** $D[u,v] \leftarrow 0$;
      else if $(t^1)_u$ or $(t^2)_v$ is a variable **then**
        $D[u,v] \leftarrow 1$;
      else /* $(t^1)_u = f_1((t^1)_{u_1}, \ldots, (t^1)_{u_p})$, $(t^2)_v = f_2((t^2)_{v_1}, \ldots, (t^2)_{v_q})$ */
        if $f_1 = f_2$ and $\langle (t^1)_{u_1}, \ldots, (t^1)_{u_p} \rangle$ can match $\langle (t^2)_{v_1}, \ldots, (t^2)_{v_q} \rangle$
        then $D[u,v] \leftarrow 1$ **else** $D[u,v] \leftarrow 0$;
  if $D[r(t_1), r(t_2)] = 1$ **then return** true **else return** false.

Step (#) takes $O(1)$ time because here $(t^1)_u$ and $(t^2)_v$ are unifiable if and only if they are the same constant or one of $(t^1)_u$ and $(t^2)_v$ is a variable.

When both u and v are internal nodes, we need to check if $\langle (t^1)_{u_1}, \ldots, (t^1)_{u_p} \rangle$ and $\langle (t^2)_{v_1}, \ldots, (t^2)_{v_q} \rangle$ can be matched. This may be done efficiently by regarding the two sequences as strings and applying string matching with variable-length don't-care symbols [4], while setting the difference to 0 and allowing don't-care symbols in both strings. Here, $(t^1)_{u_i}$ (resp., $(t^2)_{v_j}$) is regarded as a don't-care symbol that can match any substring of length at least 1 if it is a variable, otherwise $(t^1)_{u_i}$ can match $(t^2)_{v_j}$ if and only if $D[u_i, v_j] = 1$ (details are omitted in this version). It is to be noted that a variable in a term may partially match two variables in the other term. For example, consider the two terms $f(t_1, x, t_2)$ and $f(y, z)$. Here, $\theta = \{x/f(t_3, t_4),\ y/f(t_1, t_3),\ z/f(t_4, t_2)\}$ is a unifier. However, in this case, a simpler unifier is $\theta' = \{x/t_3,\ y/f(t_1, t_3),\ z/t_2\}$ because each variable occurs only once. Therefore, we can use approximate string matching with variable-length don't-care symbols, which also shows the correctness of the algorithm.

The **for**-loops are iterated $O(mn)$ times and string matching with variable-length don't-care symbols takes polynomial time, so we obtain:

Theorem 6. *Associative unification for DO-terms takes polynomial time.*

5 Commutative Unification

A function f with arity 2 is called *commutative* if $f(x, y) = f(y, x)$ always holds. Commutative unification is a variant of unification in which functions are allowed to be commutative. Commutative matching was shown to be NP-hard in [5] (by another reduction from 3SAT than the one referred to above).

First note that commutative unification is easy to solve when both t_1 and t_2 are variable-free because in this case, it reduces to the rooted unordered labeled tree isomorphism problem which is solvable in linear time (see, e.g., p. 86 in [1]):

Proposition 3. *Commutative unification for variable-free terms can be done in linear time.*

Next, we consider commutative matching. We will show how to construct a 0-1 table $D[u, v]$ for all node pairs $(u, v) \in N(t_1) \times N(t_2)$, such that $D[u, v] = 1$ if and only if $(t_1)_u$ and $(t_2)_v$ are unifiable, by applying bottom-up dynamic programming. It is enough to compute these table entries for pairs of nodes with the same depth only. We also construct a table $\Theta[u, v]$, where each entry holds a set of possible substitutions θ such that $(t_1)_u\theta = (t_2)_v$.

Let $\theta_1 = \{x_{i_1}/t_{i_1}, \ldots, x_{i_p}/t_{i_p}\}$ and $\theta_2 = \{x_{j_1}/t_{j_1}, \ldots, x_{j_p}/t_{j_q}\}$ be substitutions. θ_1 is said to be *compatible* with θ_2 if there exists no variable x such that $x = x_{i_a} = x_{j_b}$ but $t_{i_a} \neq t_{j_b}$. Let Θ_1 and Θ_2 be sets of substitutions. We define $\Theta_1 \bowtie \Theta_2 = \{\theta_i \cup \theta_j : \theta_i \in \Theta_1 \text{ is compatible with } \theta_j \in \Theta_2\}$. For any node u, u_L and u_R denote the left and right child of u. The algorithm is as follows:

Procedure $CommutMatch(t_1, t_2)$
  for all pairs $(u, v) \in N(t_1) \times N(t_2)$ with the same depth
  do /* in bottom-up order */
    if $(t_1)_u$ is a variable **then**
      $\Theta[u, v] \leftarrow \{\{(t_1)_u/(t_2)_v\}\}$; $D[u, v] \leftarrow 1$
    else if $(t_1)_u$ does not contain any variables **then**
      $\Theta[u, v] \leftarrow \emptyset$;
      if $(t_1)_u = (t_2)_v$ **then** $D[u, v] \leftarrow 1$ **else** $D[u, v] \leftarrow 0$
    else if $\gamma((t_1)_u) \neq \gamma((t_2)_v)$ **then**
      $\Theta[u, v] \leftarrow \emptyset$; $D[u, v] \leftarrow 0$ /* recall: $\gamma(t)$ is a function symbol of $r(t)$ */
    else
      $\Theta[u, v] \leftarrow \emptyset$; $D[u, v] \leftarrow 0$;
      for all $(u_1, u_2, v_1, v_2) \in \{(u_L, u_R, v_L, v_R), (u_R, u_L, v_L, v_R)\}$ **do** (#)
        if $D[u_1, v_1] = 1$ and $D[u_2, v_2] = 1$ and $\Theta_1[u_1, v_1] \bowtie \Theta_2[u_2, v_2] \neq \emptyset$
        then $\Theta[u, v] \leftarrow \Theta[u, v] \cup (\Theta_1[u_1, v_1] \bowtie \Theta_2[u_2, v_2])$; $D[u, v] \leftarrow 1$;
  if $D[r(t_1), r(t_2)] = 1$ **then return** true **else return** false.

Let B_i denote the maximum size of $\Theta[u, v]$ when the number of (distinct) variables in $(t_1)_u$ is i. Then, we have the following conjecture.

Conjecture 1. $B_1 = 1$ and $B_{i+j} = 2B_iB_j$ hold, from which $B_i = 2^{i-1}$ follows.

Theorem 7. *If Conjecture 1 holds, commutative matching can be done using* $O(2^k poly(m,n))$ *time, where* k *is the number of variables in* t_1.

Proof. The correctness follows from the observation that each variable is substituted by a term without variables and the property $f(x,y) = f(y,x)$ is taken into account at step (#). As for the time complexity, first consider the number of elements in $\Theta[u,v]$. A crucial observation is that if $(t_1)_{u_L}$ does not contain a variable then $|\Theta[u,v]| \leq \max(|\Theta[u_R,v_L]|, |\Theta[u_R,v_R]|)$ holds (and analogously for $(t_1)_{u_R}$). Assuming that Conjecture 1 is true, $\Theta_1[u_1,v_1] \bowtie \Theta_2[u_2,v_2]$ can be computed in $O(2^k poly(m,n))$ time by using 'sorting' as in usual 'join' operations. Thus, the total running time is also $O(2^k poly(m,n))$. □

Finally, we consider the case where both t_1 and t_2 contain variables. As in [21], we represent two variable-free terms t_1 and t_2 by a directed acyclic graph (DAG) $G(V,E)$, where t_1 and t_2 respectively correspond to r_1 and r_2 of indegree 0 ($r_1, r_2 \in V$). Then, testing whether r_1 and r_2 represent the same term takes polynomial time (in the size of G) by using the following procedure, where t_u denotes the term corresponding to a node u in G:

```
Procedure TestCommutIdent(r1, r2, G(V, E))
  for all u ∈ V do                                   /* in post-order */
    for all v ∈ V do                                 /* in post-order */
      if u = v then D[u, v] ← 1; continue;
      if t_u or t_v is a constant then
        if t_u = t_v then D[u, v] ← 1 else D[u, v] ← 0;
      else
        Let u = f1(u_L, u_R) and v = f2(v_L, v_R);
        if f1 = f2 then
          if (D[u_L, v_L] = 1 and D[u_R, v_R] = 1) or
                (D[u_L, v_R] = 1 and D[u_R, v_L] = 1)
          then D[u, v] ← 1 else D[u, v] ← 0
        else D[u, v] ← 0;
  if D[r1, r2] = 1 then return true else return false.
```

To cope with terms involving variables, we need to consider all possible mappings from the set of variables to $N(t_1) \cup N(t_2)$. For each such mapping, we replace all appearances of the variables by the corresponding nodes, resulting in a DAG to which we apply $TestCommutIdent(r_1, r_2, G(V,E))$. The following pseudocode describes the procedure for terms with variables:

```
Procedure CommutUnify(t1, t2)
  for all mappings M from a set of variables to nodes in t1 and t2 do
    if there exists a directed cycle (excluding a self-loop) then continue;
    Replace each variable having a self-loop with a distinct constant symbol;
    Replace each occurrence of a variable node u with node M(u);
          /* if M(u) = v and M(v) = w then u is replaced by w */
    Let G(V, E) be the resulting DAG;
```

Let r_1 and r_2 be the nodes of G corresponding to t_1 and t_2;
if $TestCommutIdent(r_1, r_2, G(V, E)) = \text{true}$ **then return** true;
return false.

In summary, we have the following theorem, which implies that commutative unification belongs to the class XP [12].

Theorem 8. *Commutative unification can be done in* $O((m+n)^{k+2})$ *time.*

Proof. The correctness of $TestCommutIdent(r_1, r_2, G(V, E))$ follows from the fact that $f_1(t_1, t_2)$ matches $f_2(t'_1, t'_2)$ if and only if f_1 and f_2 are identical function symbols and either (t_1, t_2) matches (t'_1, t'_2) or (t_1, t_2) matches (t'_2, t'_1). It is clear that this procedure runs in $O(mn)$ time. Therefore, commutative matching of two variable-free terms can be done in polynomial time.

Next, we consider $CommutUnify(t_1, t_2)$. For an illustration of how it works, see Fig. 3. To prove the correctness, it is straightforward to see that if there exists some mapping M which returns 'true', then t_1 and t_2 are commutatively unifiable and such a mapping gives a substitution θ satisfying $t_1\theta = t_2\theta$. Conversely, suppose that t_1 and t_2 are commutatively unifiable. Then there exist unifiable non-commutative terms t'_1 and t'_2 that are obtained by exchanging the left and right arguments in some terms in t_1 and t_2. Let θ be the substitution satisfying $t'_1\theta = t'_2\theta$. Then, $t_1\theta = t_2\theta$ holds. We assign distinct constants to variables appearing in $t_1\theta$. We also construct a mapping from the remaining variables to $N(t_1) \cup N(t_2)$ by regarding $x/t \in \theta$ as a mapping of x to t. We construct $G(V, E)$ according to this mapping. Then, it is obvious that $TestCommutIdent(r_1, r_2, G(V, E)) = \text{true}$ holds.

Since the number of possible mappings is bounded by $(m+n)^k$, where k is the number of variables in t_1 and t_2, $CommutUnify(t_1, t_2)$ runs in $O((m+n)^{k+2})$ time. □

6 Associative-Commutative Unification

Associative-commutative unification is the variant of unification in which some functions can be both associative and commutative. The next theorem, whose proof is omitted in this version, shows that associative-commutative matching is $W[1]$-hard even if every function is associative and commutative.

Theorem 9. *Matching is* $W[1]$*-hard with respect to the number of variables even if every function symbol is associative and commutative.*

On the other hand, associative-commutative matching can be done in polynomial time if t_1 is a DO-term [5]. We can extend this algorithm to the special case of unification where both terms are DO-terms by adding a condition in the algorithm that $f((t_1)_{u_1}, \ldots, (t_1)_{u_p})$ and $f((t_2)_{v_1}, \ldots, (t_2)_{v_q})$ can be unified if $(t_1)_{u_i}$ and $(t_2)_{v_j}$ are variables for some i, j. This yields:

Proposition 4. *Associative-commutative unification can be done in polynomial time if both* t_1 *and* t_2 *are DO-terms.*

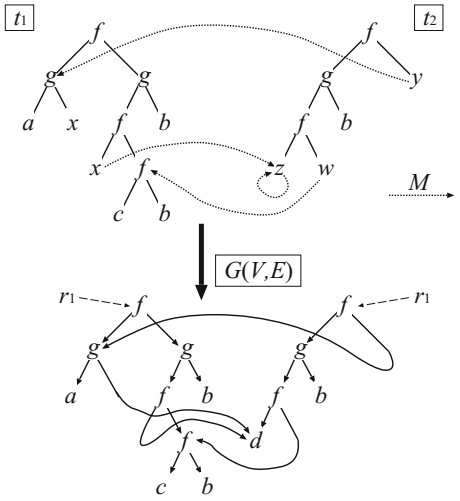

Fig. 3. Example of a DAG $G(V, E)$ for $CommutIdent$ and for the proof of Theorem 8.

7 Concluding Remarks

This paper has studied the parameterized complexity of unification with associative and/or commutative functions with respect to the number of variables. Determining whether each of commutative unification and the matching version of Theorem 2 (i.e., where all variables occur in one of the strings and the number of occurrences of each variable is at most 2), is $W[1]$-hard or FPT and whether associative unification is in XP, as well as any nontrivial improvements of the presented results, are left as open problems.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Aikou, K., Suzuki, Y., Shoudai, T., Uchida, T., Miyahara, T.: A polynomial time matching algorithm of ordered tree patterns having height-constrained variables. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 346–357. Springer, Heidelberg (2005)
3. Akutsu, T.: Approximate string matching with don't care characters. Inf. Process. Lett. **55**, 235–239 (1995)
4. Akutsu, T.: Approximate string matching with variable length don't care characters. IEICE Trans. Inf. Syst. **E79–D**, 1353–1354 (1996)
5. Benanav, D., Kapur, D., Narendran, P.: Complexity of matching problems. J. Symbolic Comput. **3**, 203–216 (1987)
6. Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 447–533. Elsevier, Amsterdam (2001)
7. Bille, P.: A survey on tree edit distance and related problem. Theoret. Comput. Sci. **337**, 217–239 (2005)

8. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Wareham, H.T.: The parameterized complexity of sequence alignment and consensus. Theoret. Comput. Sci. **147**, 31–54 (1995)
9. de Champeaux, D.: About the paterson-wegman linear unification algorithm. J. Comput. Syst. Sci. **32**, 79–90 (1986)
10. Eker, S.: Single elementary associative-commutative matching. J. Autom. Reasoning **28**, 35–51 (2002)
11. Fernau, H., Schmid, M.L.: Pattern matching with variables: A multivariate complexity analysis. In: Fischer, J., Sanders, P. (eds.) CPM 2013. LNCS, vol. 7922, pp. 83–94. Springer, Heidelberg (2013)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
13. Gilbert, D., Schroeder, M.: FURY: fuzzy unification and resolution based on edit distance. In: Proceedings of 1st IEEE International Symposium on Bioinformatics and Biomedical Engineering, pp. 330–336 (2000)
14. Iranzo, P.J., Rubio-Manzano, C.: An efficient fuzzy unification method and its implementation into the Bousi∼Prolog system. In: Proceedings of 2010 IEEE International Conference on Fuzzy Systems, pp. 1–8 (2010)
15. Jones, N.C., Pevzner, P.A.: An Introduction to Bioinformatics Algorithms. MIT Press, Cambridge (2004)
16. Kamali, S., Tompa, F.W.: A new mathematics retrieval system. In: Proceedings of ACM Conference on Information and Knowledge Management, pp. 1413–1416 (2010)
17. Kapur, D., Narendran, P.: Complexity of unification problems with associative-commutative operators. J. Autom. Reasoning **28**, 35–51 (2002)
18. Knight, K.: Unification: a multidisciplinary survey. ACM Comput. Surv. **21**, 93–124 (1989)
19. Nguyen, T.T., Chang, K., Hui, S.C.: A math-aware search engine for math question answering system. In: Proceedings of ACM Conference on Information and Knowledge Management, pp. 724–733 (2012)
20. NTCIR: http://research.nii.ac.jp/ntcir/ntcir-10/conference.html (2013)
21. Paterson, M.S., Wegman, M.N.: Linear unification. J. Comput. Syst. Sci. **16**, 158–167 (1978)
22. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM **12**, 23–41 (1965)

http://www.springer.com/978-3-319-13523-6

Parameterized and Exact Computation
9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers
Cygan, M.; Heggernes, P. (Eds.)
2014, IX, 343 p. 37 illus., Softcover
ISBN: 978-3-319-13523-6