Article Submitted to Journal of Symbolic Computation

Tractable and Intractable Second-Order Matching Problems *†

Kouichi Hirata[1], Keizo Yamada[1] and Masateru Harao[1]

[1]*Department of Artificial Intelligence, Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan.*‡

Abstract

The *second-order matching problem* is to determine whether or not a first-order term without variables is an instance of a second-order term that is allowed to contain not only individual variables but also function variables. It is well-known that the second-order matching problem is NP-complete in general. In this paper, we first introduce the several restrictions for the second-order matching problems, such as the bounded number, arity and occurrence of function variables, ground that contains no individual variables, flat that contains no function constants, and predicate that no function variable occurs in the terms of arguments of each function variable. By combining the above restrictions, we give the sharp separations of tractable second-order matching problems from intractable ones. Finally, we compare them with the separations of decidable second-order unification problems from undecidable ones.

1. Introduction

The *unification problem* is to determine whether or not any two terms possess a common instance. The *matching problem*, on the other hand, is to determine whether or not a term is an instance of another term. Both the unification and matching play an important role in many research areas, including resolution-based theorem proving, term rewriting systems, logic and functional

*The preliminary version of this paper was published in *Proceedings of the 5th Annual International Computing and Combinatorics Conference*, Lecture Notes in Computer Science 1627, Springer-Verlag, 1999, pp.432–442.

†This work is partially supported by the Japan Society for the Promotion of Science, Grand-in-Aid for Scientific Research (B) 13558036 and for Encouragement of Young Scientists (B) 15700137.

‡email: {hirata,yamada,harao}@ai.kyutech.ac.jp

programming, constraint-based programming, program synthesis and transformation, database query language, and so on.

For *higher-order unification* and *matching problems*, it is known the following general complexity results. The higher-order unification problem is undecidable even if the order is 2 [14]. On the other hand, the higher-order matching problem is decidable for the order ≤ 4 and remains open for the order ≥ 5 [3, 28]. In particular, both the second- and third-order matching problems are NP-complete [2, 3, 12, 28].

According to [10, 14], let L be a *term language* consisting of individual constants, individual variables, function constants and function variables. Then, *L-terms* are defined similar as first-order terms allowing function variables.

The *second-order unification problem in L* is formulated as the problem of determining, for a finite set $\{\langle t_i, s_i\rangle \mid i \in I\}$ of pairs of L-terms t_i and s_i, called a *unification expression in L*, whether or not there exists a substitution σ such that $t_i\sigma = s_i\sigma$ for each $i \in I$. The *second-order matching problem in L* is the special second-order unification problem in L that s_i contains no variables. Hence, it is formulated as the problem of determining whether or not there exists a substitution σ such that $t_i\sigma = s_i$ for each $i \in I$. We call a unification expression in L for the second-order matching problem a *matching expression in L*.

Concerned with the second-order matching problem, Huet [17] has designed a complete and nonredundant algorithm based on the transformation rule consisting of a *simplification*, an *imitation*, and a *projection*. Intuitively, the reason why the second-order matching problem is intractable is to allow the projection.

Since the second-order unification problem is undecidable as mentioned above, various researchers have separated decidable unification problems from undecidable ones, by introducing the several restrictions for term languages or unification expressions [1, 8, 9, 10, 13, 14, 21, 22, 23, 26]. On the other hand, while the second-order matching problem is NP-complete [2], there exist few researches to analyze deeply the complexity of the matching problem as similar as the unification problems. It is one of the reasons that the interest of the researchers is rather to design the matching algorithm aiming to apply to program synthesis and transformation, schema-guided proof, analogical reasoning and machine learning [5, 6, 7, 11, 15, 18, 19, 29], than the matching problem itself.

In this paper, by introducing the following restrictions for term languages and matching expressions, we give the several sharp separations of tractable second-order matching problems from intractable ones.

A term language L is called *k-ary* if each function variable in L is at most k-ary and *k-fv* if L contains at most k distinct function variables. Also a term language L is called *flat* if L contains no function constants, *ground* if L contains no individual variables, *monadic* [9] if each function constant in L is unary, and *nonmonadic* [10] if L is not monadic. On the other hand, a matching expression E is called *predicate* if no function variable occurs in terms of any argument of each function variable in E and *read-k-times* if each function variable in E occurs at most k times.

Hence, we obtain the following results:

1. The second-order matching problems remain NP-complete for read-twice predicate matching expressions in unary or 1-fv term languages, for read-thrice predicate matching expressions in ternary flat term languages, and for arbitrary matching expressions in unary ground or binary flat ground term languages.
2. As the corollaries, the second-order matching problems also remain NP-complete for arbitrary matching expressions in nonmonadic unary or monadic term languages, and for read-twice predicate matching expressions.
3. On the other hand, the second-order matching problems are solvable in polynomial time for predicate matching expressions in binary flat or ground term languages, and for arbitrary matching expressions in unary flat or k-fv ($k \geq 0$) flat term languages.

Finally, we compare the above results with the separations of decidable second-order unification problems from undecidable ones given by [1, 8, 9, 10, 13, 14, 21, 22, 23, 26].

2. Preliminaries

Instead of considering arbitrary second-order languages, we shall restrict our attention to languages containing just simple terms (i.e., terms without variable-binding operators like the λ operator, e.g., [16, 17, 18, 27]). Throughout this paper, we deal with the term languages used by Goldfarb [14] and Farmer [10].

Let a *term language* L be a quadruple $(\mathrm{IC}_L, \mathrm{IV}_L, \mathrm{FC}_L, \mathrm{FV}_L)$, where

1. IC_L is a set of *individual constants* (denoted by $a, b, c, \ldots$);
2. IV_L is a set of *individual variables* (denoted by $x, y, z, \ldots$);
3. FC_L is a set of *function constants* (denoted by $f, g, h, \ldots$);
4. FV_L is a set of *function variables* (denoted by $F, G, H, \ldots$).

Each element of $\mathrm{FC}_L \cup \mathrm{FV}_L$ has a fixed arity ≥ 1, and IC_L, IV_L, FC_L and FV_L are mutually disjoint. We call an element of $\mathrm{IV}_L \cup \mathrm{FV}_L$ a *variable* simply.

The *L-terms* are defined inductively by:

1. Each $d \in \mathrm{IC}_L \cup \mathrm{IV}_L$ is an L-term.
2. If $d \in \mathrm{FC}_L \cup \mathrm{FV}_L$ has an arity $n \geq 1$ and $t_1, \ldots, t_n$ are L-terms, then $d(t_1, \ldots, t_n)$ is an L-term.

Let BV_L be an infinite collection $\{w_i\}_{i \geq 1}$ of symbols not contained in L called *bound variables*. Then, the *L^*-terms* are defined inductively by:

1. Each $d \in \mathrm{IC}_L \cup \mathrm{IV}_L \cup \mathrm{BV}_L$ is an L^*-term.
2. If $d \in \mathrm{FC}_L \cup \mathrm{FV}_L$ has an arity $n \geq 1$ and $t_1, \ldots, t_n$ are L^*-terms, then $d(t_1, \ldots, t_n)$ is an L^*-term.

The *rank* of an L^*-term t is the largest n such that w_n occurs in t. (L-terms have rank 0.) For $n \geq 1$, L^*-terms of rank n intuitively represent n-ary functions. We sometimes call an L-term simply a *term*.

Let t be an L^*-term. The *head* of t, denoted by $\mathrm{hd}(t)$, is the outermost symbol occurring in t. We say that t is *closed* if t contains no variables. The *size* of t, denoted by $|t|$, is the number of symbols of L occurring in t.

For L^*-terms $t, t_1, \ldots, t_n$, we write $t[t_1, \ldots, t_n]$ for the L^*-term obtained by replacing each occurrence of w_i in t with t_i for all i $(1 \leq i \leq n)$ simultaneously.

A *substitution* in L is a function σ with a finite domain $\mathrm{dom}(\sigma) \subseteq \mathrm{IV}_L \cup \mathrm{FV}_L$ which maps individual variables to L-terms and n-ary function variables with $n \geq 1$ to L^*-terms of rank $\leq n$. The result applying a substitution σ in L to $v \in \mathrm{dom}(\sigma)$ is denoted by $v\sigma$ instead of $\sigma(v)$. We assume that $x\sigma \neq x$ and $F\sigma \neq F(w_1, \ldots, w_n)$ for all substitutions σ and $x, F \in \mathrm{dom}(\sigma)$. A substitution σ is denoted as $\{s_1/v_1, \ldots, s_m/v_m\}$, where $\mathrm{dom}(\sigma) = \{v_1, \ldots, v_m\}$ and σ maps v_i to s_i for each i $(1 \leq i \leq m)$. Each s_i/v_i is called a *binding* of σ.

Let σ be a substitution $\{s_1/v_1, \ldots, s_m/v_m\}$ in L. The result $t\sigma$ of applying σ to an L^*-term t is defined inductively by:

1. If $t = c$, then $t\sigma = c$.
2. If $t = x$ and $x \in \mathrm{dom}(\sigma)$, then $t\sigma = x\sigma$.
3. If $t = x$ and $x \notin \mathrm{dom}(\sigma)$, then $t\sigma = x$.
4. If $t = f(t_1, \ldots, t_n)$, then $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.
5. If $t = F(t_1, \ldots, t_n)$ and $F \notin \mathrm{dom}(\sigma)$, then $t\sigma = F(t_1\sigma, \ldots, t_n\sigma)$.
6. If $t = F(t_1, \ldots, t_n)$ and $F \in \mathrm{dom}(\sigma)$, then $t\sigma = (F\sigma)[t_1\sigma, \ldots, t_n\sigma]$.

The *composition of substitutions* σ and θ, denoted by $\sigma\theta$, is the substitution such that $v(\sigma\theta) = (v\sigma)\theta$ for any variable v.

A *matching expression in* L is any finite set $E = \{\langle t_i, s_i\rangle \mid i \in I\}$, where t_i is an L-term and s_i is a closed L-term for each $i \in I$. For a substitution σ, $E\sigma$ denotes the matching expression $\{\langle t_i\sigma, s_i\rangle \mid i \in I\}$. For a matching expression E and a function variable F, E_F denotes a matching expression $\{\langle t, s\rangle \in E \mid \mathrm{hd}(t) = F\}$. The *size* of E, denoted by $|E|$, is defined to be $\sum_{i\in I}(|t_i| + |s_i|)$.

Let L be a term language and E be a matching expression $\{\langle t_i, s_i\rangle \mid i \in I\}$ in L. Then, E is *matchable in* L if there exists a substitution σ in L such that $t_i\sigma = s_i$ for each $i \in I$. Such a substitution is called a *matcher* of E in L, and we also say that σ *matches* E in L. We sometimes omit the notion "in L".

Huet [17] has designed the *second-order matching algorithm*, based on the following transformation rule.

Definition (Huet [17]): Let E be a matching expression in L. Then, the *transformation rule* $\Rightarrow$ is defined as follows:

1. **simplification**:

$$\{\langle f(t_1, \ldots, t_n), f(s_1, \ldots, s_n)\rangle\} \cup E \Rightarrow \{\langle t_1, s_1\rangle, \ldots, \langle t_n, s_n\rangle\} \cup E \quad (n \geq 0).$$

2. **imitation** (on F): if $\langle F(t_1,\ldots,t_n), f(s_1,\ldots,s_m)\rangle \in E$ $(n,m \geq 0)$, then
$$E \Rightarrow E\{f(H_1(w_1,\ldots,w_n),\ldots,H_m(w_1,\ldots,w_n))/F\}.$$
Here, $H_1,\ldots,H_m$ are new function variables not occurring in E.
3. **projection** (on F): if $\langle F(t_1,\ldots,t_n), s\rangle \in E$ $(n \geq 1, 1 \leq i \leq n)$, then
$$E \Rightarrow E\{w_i/F\}.$$

By a simplification that $n = 0$, a matching expression $\{\langle c, c\rangle\} \cup E$ $(c \in \mathrm{IC}_L)$ is transformed to E. Furthermore, if E contains no function variables, then a projection cannot be applied to E but an imitation can as $E \Rightarrow E\{s/x\}$ if $\langle x, s\rangle \in E$ and $x \in \mathrm{IV}_L$.

The transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. Then, the following theorem holds:

Theorem 2.1 (Huet [17]): *A matching expression E is matchable in L if and only if $E \Rightarrow^* \emptyset$.*

By Theorem 2.1, we can obtain the following corollary.

Corollary 2.1: *Let L be a term language and E be a matching expression in L such that $E_F = \{\langle F(t_1^i,\ldots,t_n^i), f(s_1^i,\ldots,s_m^i)\rangle \mid i \in I\}$ for $F \in \mathrm{FV}_L$. Also let L' and E' be the following term language and matching expression in L', respectively:*
$$\begin{aligned} L' &= (\mathrm{IC}_L, \mathrm{IV}_L, \mathrm{FC}_L, \mathrm{FV}_L \cup \{H_1,\ldots,H_m\}),\\ E' &= E\{f(H_1(w_1,\ldots,w_n),\ldots,H_m(w_1,\ldots,w_n))/F\}. \end{aligned}$$
Then, E is matchable in L if and only if either of the following two statements holds:

1. *E' is matchable in L', or*
2. *there exists an index j $(1 \leq j \leq n)$ and a substitution σ in L such that $t_j^i\sigma = f(s_1^i,\ldots,s_m^i)$ for each $i \in I$.*

The main topic in this paper is to analyze the computational complexity of the following *second-order matching problem in a term language L*:

Second-Order Matching (Matching)

Instance: A matching expression E in a term language L.

Question: Is E matchable in L?

The following theorem has been shown in early 1970's.

Theorem 2.2 (Baxter [2, 12]): Matching *is NP-complete.*

3. Complexity of the Second-Order Matching Problems

First we introduce the following restrictions for term languages and matching expressions.

Definition: Let L be a term language.

1. L is *k-ary* if each function variable in L is at most k-ary. For the case that $k = 1$, 2 or 3, we call it *unary*, *binary* or *ternary*, respectively.
2. L is *k-fv* if L contains at most k distinct function variables, that is, $\#\mathrm{FV}_L \leq k$. (Here, # denotes the cardinality of a set.)
3. L is *ground* if L contains no individual variables, that is, $\mathrm{IV}_L = \emptyset$.
4. L is *flat* if L contains no function constants, that is, $\mathrm{FC}_L = \emptyset$.
5. L is *monadic* if each function constant in L is unary.
6. L is *nonmonadic* if L is not monadic.

Definition: Let E be a matching expression in L.

1. E is *read-k-times* if each function variable occurs in E at most k times. For the case that $k = 1$, 2 or 3, we call it *read-once*, *read-twice* or *read-thrice*, respectively.
2. E is *predicate* if no function variable occurs in the terms of arguments of each function variable in E.

Then, we formulate the following restricted problems for Matching:

kAry (*resp.*, kFV, Ground, Flat, Mon, Nonmon) Matching

Instance: A matching expression E in a k-ary (*resp.*, k-fv, ground, flat, monadic, nonmonadic) term language L.

Question: Is E matchable in L?

kTimes (*resp.*, Pred) Matching

Instance: A read-k-times (*resp.*, predicate) matching expression E in a term language L.

Question: Is E matchable in L?

In this section, we analyze the computational complexity of Matching, by combining the above restrictions. In order to show the tractability, we either design the deterministic applications of the transformation rule in Definition 2.1 (Theorem 3.6, 3.8 and 3.9) or reduce it to the tractable problem (Theorem 3.4).

In order to show the intractability, on the other hand, we reduce the following problem Monotone 1-in-3 3SAT [12] to the several restricted problems for Matching (Theorem 3.1, 3.2, 3.3, 3.5 and 3.7). It is said that this reduction technique is also a key idea of *word unification* or *word matching problem*.

Monotone 1-in-3 3SAT

INSTANCE: A set X of variables and a collection C of monotone 3-clauses (i.e., clauses consisting of exactly three positive literals) over X.

QUESTION: Is there a truth assignment to X that makes exactly one literal of each clause in C true?

Throughout this paper, we fix X and C to the set $\{x_1, \ldots, x_n\}$ of variables and the set $\{c_1, \ldots, c_m\}$ of clauses, respectively, as an instance of MONOTONE 1-IN-3 3SAT. In particular, we assume that $c_j \in C$ consists of variables x_1^j, x_2^j and x_3^j for each j $(1 \leq j \leq m)$, and (nonindexed) $c \in C$ consists of variables z_1, z_2 and z_3.

By Theorem 2.2, the restricted problems for MATCHING to be treated below are always in NP, so we do not state it explicitly.

3.1. The bounded arity of function variables

In this section, we investigate the computational complexity of the restricted problems for kARYMATCHING.

THEOREM 3.1: UNARYTWICEPREDMATCHING *is NP-complete. In other words,* MATCHING *is NP-complete even if*

1. *each function variable is unary and occurs at most twice, and*
2. *no function variable occurs below other function variables.*

Proof: Let C be an instance of MONOTONE 1-IN-3 3SAT over X. Consider the following unary term language L:

$$L = (\{0, 1\}, X \cup \{y_1, \ldots, y_m\}, \{f\}, \{F_1, \ldots, F_m\}).$$

Here, F_j $(1 \leq j \leq m)$ is a unary function variable and f is a binary function constant. For each clause $c = z_1 \vee z_2 \vee z_3 \in C$, let E_c be the following read-twice predicate matching expression:

$$E_c = \left\{ \begin{array}{ll} \langle F(f(z_3, f(z_2, f(z_1, y)))), & f(0, f(0, f(1, f(0, f(0, 0))))) \rangle, \\ \langle F(y), & f(0, f(0, 0)) \rangle \end{array} \right\}.$$

Suppose that c is satisfiable and let (a_1, a_2, a_3) be a truth assignment to (z_1, z_2, z_3) satisfying c, where there exists exactly one index i $(1 \leq i \leq 3)$ such that $a_i = 1$ and $a_l = 0$ $(l \neq i)$. Hence, we can construct the matcher σ of E_c as follows:

1. If $(a_1, a_2, a_3) = (1, 0, 0)$, then $\sigma = \{w_1/F, 1/z_1, 0/z_2, 0/z_3, f(0, f(0, 0))/y\}$;
2. If $(a_1, a_2, a_3) = (0, 1, 0)$, then $\sigma = \{f(0, w_1)/F, 0/z_1, 1/z_2, 0/z_3, f(0, 0)/y\}$;
3. If $(a_1, a_2, a_3) = (0, 0, 1)$, then $\sigma = \{f(0, f(0, w_1))/F, 0/z_1, 0/z_2, 1/z_3, 0/y\}$.

Conversely, suppose that E_c is matchable and let σ be a matcher of E_c. Then, σ contains the binding t/F, where t is w_1, $f(0, w_1)$, or $f(0, f(0, w_1))$.

Suppose that $w_1/F \in \sigma$. Since $E_c\{w_1/F\}$ is of the form

$$\{\langle f(z_3, f(z_2, f(z_1, y))), f(0, f(0, f(1, f(0, f(0, 0))))))\rangle, \langle y, f(0, f(0, 0))\rangle\},$$

and by a simplification, σ contains the bindings $1/z_1$, $0/z_2$ and $0/z_3$.

Suppose that $f(0, w_1)/F \in \sigma$. Since $E_c\{f(0, w_1)/F\}$ is of the form

$$\{\langle f(0, f(z_3, f(z_2, f(z_1, y)))), f(0, f(0, f(1, f(0, f(0, 0))))))\rangle, \langle y, f(0, 0)\rangle\},$$

and by a simplification, σ contains the bindings $0/z_1$, $1/z_2$ and $0/z_3$.

Suppose that $f(0, f(0, w_1))/F \in \sigma$. Since $E_c\{f(0, f(0, w_1))/F\}$ is of the form

$$\{\langle f(0, f(0, f(z_3, f(z_2, f(z_1, y))))), f(0, f(0, f(1, f(0, f(0, 0))))))\rangle, \langle y, 0\rangle\},$$

and by a simplification, σ contains the bindings $0/z_1$, $0/z_2$ and $1/z_3$.

Then, we can construct the truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) satisfying c such that $a_i = 1$ if $1/z_i \in \sigma$; $a_i = 0$ if $0/z_i \in \sigma$ $(1 \leq i \leq 3)$. Hence, (a_1, a_2, a_3) satisfies c, where exactly one of a_1, a_2 and a_3 is 1 and others are 0.

For $C = \{c_1, \ldots, c_m\}$, let E be the following read-twice predicate matching expression in L:

$$\bigcup_{j=1}^{m}(E_{c_j}\{F_j(w_1)/F, y_j/y\}).$$

Then, C is satisfiable by a truth assignment that makes exactly one literal of each clause in C true if and only if E is matchable in L. □

Theorem 3.2: UnaryGroundMatching *is NP-complete. In other words,* Matching *is NP-complete even if*

1. *each function variable is unary and*
2. *no individual variable occurs.*

Proof: Let C be an instance of Monotone 1-in-3 3SAT over X. Consider the following unary ground term language L:

$$L = (\{0, 1\}, \emptyset, \{f\}, \{F_{x_1}, \ldots, F_{x_n}\}).$$

Here, F_{x_i} $(1 \leq i \leq n)$ is a unary function variable and f be a unary function constant. For each clause $c = z_1 \vee z_2 \vee z_3 \in C$, let E_c be the following matching expression:

$$E_c = \{\langle F_{z_1}(F_{z_2}(F_{z_3}(0))), f(0)\rangle, \langle F_{z_1}(F_{z_2}(F_{z_3}(1))), f(1)\rangle\}.$$

If c is satisfiable by a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ for exactly one index i $(1 \leq i \leq 3)$ and $a_l = 0$ $(1 \leq l \leq 3, l \neq i)$, then we can construct a matcher σ of E_c such that $f(w_1)/F_{z_i} \in \sigma$ if $a_i = 1$ and $w_1/F_{z_l} \in \sigma$ if $a_l = 0$. Conversely, if E_c is matchable and σ is a matcher of E_c, then there exists exactly one index i $(1 \leq i \leq 3)$ such that $f(w_1)/F_{z_i} \in \sigma$ and $w_1/F_{z_l} \in \sigma$ $(1 \leq l \leq 3, l \neq i)$. Thus we can construct a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ if $f(w_1)/F_{z_i} \in \sigma$ and $a_l = 0$ if $w_1/F_{z_l} \in \sigma$. Hence, c is satisfiable by a truth assignment that makes exactly one literal true if and only if E_c is matchable.

For $C = \{c_1, \ldots, c_m\}$, let E be the matching expression $\bigcup_{j=1}^{m} E_{c_j}$ in L. Then, C is satisfiable by a truth assignment that makes exactly one literal of each clause in C true if and only if E is matchable in L. □

In the proofs of Theorem 3.1 and 3.2, we cannot eliminate the function constant f. Hence, the existence of such a function constant is essential for these intractabilities. On the other hand, for the flat term language, the following theorem holds.

Theorem 3.3: TernaryFlatThricePredMatching *is NP-complete. In other words,* Matching *is NP-complete even if*

1. *each function variable is at most ternary and occurs at most thrice,*
2. *no function constant occurs, and*
3. *no function variable occurs below other function variables.*

Proof: Let C be an instance of Monotone 1-in-3 3SAT over X. Consider the following ternary flat term language L:

$$L = (\{0, 1\}, X, \emptyset, \{F_1, \ldots, F_m\}).$$

Here, F_j $(1 \leq j \leq m)$ is a ternary function variable. For each clause $c = z_1 \vee z_2 \vee z_3 \in C$, let E_c be the following read-thrice predicate matching expression:

$$E_c = \{\langle F(z_1, z_2, z_3), 1\rangle, \langle F(z_2, z_3, z_1), 0\rangle, \langle F(z_3, z_1, z_2), 0\rangle\}.$$

If c is satisfiable by a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ for exactly one index i $(1 \leq i \leq 3)$ and $a_l = 0$ $(1 \leq l \leq 3, l \neq i)$, then we can construct a matcher σ of E_c such that $1/z_i \in \sigma$ if $a_i = 1$ and $0/z_l \in \sigma$ if $a_l = 0$. Conversely, if E_c is matchable and σ is a matcher of E_c, then there exists exactly one index i $(1 \leq i \leq 3)$ such that $1/z_i \in \sigma$ and $0/z_l \in \sigma$ $(1 \leq l \leq 3, l \neq i)$, because σ must contain one of the bindings w_1/F, w_2/F or w_3/F. Thus we can construct a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ if $1/z_i \in \sigma$ and $a_l = 0$ if $0/z_l \in \sigma$. Hence, c is satisfiable by a truth assignment that makes exactly one literal true if and only if E_c is matchable.

For $C = \{c_1, \ldots, c_m\}$, let E be the following read-thrice predicate matching expression in L:

$$\bigcup_{j=1}^{m}(E_{c_j}\{F_j(w_1, w_2, w_3)/F\}).$$

Then, C is satisfiable by a truth assignment that makes exactly one literal of each clause in C true if and only if E is matchable in L. □

If we strengthen the condition *ternary* in Theorem 3.3 to *binary* and weaken the condition *read-thrice* to nothing, respectively, then we obtain the following result.

THEOREM 3.4: BINARYFLATPREDMATCHING *is solvable in polynomial time. In other words,* MATCHING *is solvable in polynomial time if*

1. *each function variable is at most binary,*
2. *no function constant occurs, and*
3. *no function variable occurs below other function variables.*

Proof: We reduce BINARYFLATPREDMATCHING to 2SAT [12], which is solvable in polynomial time:

2SAT

INSTANCE: A set X of variables and a collection C of 2-clauses (i.e., clauses consisting of at most two literals) over X.

QUESTION: Is there a truth assignment to X satisfying C?

Let L be a binary flat term language and E be a predicate matching expression in L. If E_F is of the form $\{\langle t_i, s\rangle \mid i \in I\}$, then E_F is always matchable. Hence, without loss of generality, we can suppose that, for a function variable F, E_F contains pairs $\langle t_1, s_1\rangle$ and $\langle t_2, s_2\rangle$ such that $s_1 \neq s_2$.

Let IC_E, IV_E, and FV_E be the sets of all individual constants, individual variables, and function variables in E, respectively. Then, E_F is of the form either (1) $\{\langle F(t_{i,1}, t_{i,2}), s_i\rangle \mid i \in I, s_i \in \mathrm{IC}_L\}$ or (2) $\{\langle F(t_{i,1}), s_i\rangle \mid i \in I, s_i \in \mathrm{IC}_L\}$.

In Case (1), for each pair $\langle F(t_{i,1}, t_{i,2}), s_i\rangle \in E_F$, construct the following formula $T^F_{i,j}$ $(j = 1, 2)$:

1. If $t_{i,j} \in \mathrm{IC}_E$ and $t_{i,j} = s_i$, then $T^F_{i,j} = \mathtt{true}$;
2. If $t_{i,j} \in \mathrm{IC}_E$ and $t_{i,j} \neq s_i$, then $T^F_{i,j} = \mathtt{false}$;
3. If $t_{i,j} = v \in \mathrm{IV}_E$, then $T^F_{i,j} = x_{vs_i} \wedge (\bigwedge_{c \in \mathrm{IC}_E - \{s_i\}} \overline{x_{vc}})$.

Also let T_{E_F} be a DNF formula $(\bigwedge_{i\in I} T^F_{i,1}) \vee (\bigwedge_{i\in I} T^F_{i,2})$. In Case (2), for each pair $\langle F(t_{i,1}), s_i\rangle \in E_F$, construct the formula $T^F_{i,1}$ as above, and let T_{E_F} be $\bigwedge_{i\in I} T^F_{i,1}$.

For example, let E be the following expression:

$$\left\{ \begin{array}{llll} \langle F(v, y), a\rangle, & \langle F(b, v), b\rangle, & \langle F(y, z), b\rangle, & \langle F(z, c), c\rangle, \\ \langle G(y, v), a\rangle, & \langle G(b, y), b\rangle & & \end{array} \right\}.$$

Then, T_{E_F} and T_{E_G} are constructed as follows:

$$
\begin{aligned}
T_{E_F} \;=\; \{(&\underbrace{(x_{va} \wedge \overline{x_{vb}} \wedge \overline{x_{vc}})}_{T^F_{1,1}} \wedge \underbrace{\mathtt{true}}_{T^F_{2,1}} \wedge \underbrace{(x_{yb} \wedge \overline{x_{ya}} \wedge \overline{x_{yc}})}_{T^F_{3,1}} \wedge \underbrace{(x_{zc} \wedge \overline{x_{zb}} \wedge \overline{x_{zb}})}_{T^F_{4,1}} \\
&\vee \underbrace{(x_{ya} \wedge \overline{x_{yb}} \wedge \overline{x_{yc}})}_{T^F_{1,2}} \wedge \underbrace{(x_{vb} \wedge \overline{x_{va}} \wedge \overline{x_{vc}})}_{T^F_{2,2}} \wedge \underbrace{(x_{zb} \wedge \overline{x_{za}} \wedge \overline{x_{zc}})}_{T^F_{3,2}} \wedge \underbrace{\mathtt{true}}_{T^F_{4,2}})\} \\
T_{E_G} \;=\; \{(&\underbrace{(x_{ya} \wedge \overline{x_{yb}} \wedge \overline{x_{yc}})}_{T^G_{1,1}} \wedge \underbrace{\mathtt{true}}_{T^G_{2,1}}) \;\vee\; (\underbrace{(x_{va} \wedge \overline{x_{vb}} \wedge \overline{x_{vc}})}_{T^G_{1,2}} \wedge \underbrace{(x_{yb} \wedge \overline{x_{ya}} \wedge \overline{x_{yc}})}_{T^G_{2,2}})\}
\end{aligned}
$$

The 2CNF formula equivalent to T_{E_F} is denoted by C_{E_F} and $\bigwedge_{F \in \mathrm{FV}_E} C_{E_F}$ is denoted by C_E. Note that the number of clauses in C_E is at most $(\#\mathrm{IC}_E \times \#E)^2 \times \#\mathrm{FV}_E \leq |E|^5$.

Suppose that C_E is satisfiable and let a be a truth assignment to variables $\{x_{vc} \mid v \in \mathrm{IV}_E, c \in \mathrm{IC}_E\}$ satisfying C_E. By the definition of C_E, a satisfies C_{E_F} for each $F \in \mathrm{FV}_E$, so it satisfies T_{E_F} for each $F \in \mathrm{FV}_E$. Then, it also satisfies $\bigwedge_{i \in I} T^F_{i,1}$, $\bigwedge_{i \in I} T^F_{i,2}$, or both for each $F \in \mathrm{FV}_E$. If a satisfies $\bigwedge_{i \in I} T^F_{i,j}$ $(j = 1, 2)$, then we add the bindings w_j/F and c/v to σ for each positive literal $x_{vc} \in \bigwedge_{i \in I} T^F_{i,j}$ for each $F \in \mathrm{FV}_E$. Hence, by the construction of σ and the definition of $\bigwedge_{i \in I} T^F_{i,j}$, σ is a matcher of E.

In the above example, let a be a truth assignment that assigns 1 to x_{ya}, x_{vb} and x_{zb} and 0 to the other variables. Then, a satisfies C_E, and σ is constructed as $\{w_2/F, w_1/G, a/y, b/v, b/z\}$.

Conversely, suppose that E is matchable and let σ be a matcher of E. For $v \in \mathrm{IV}_E$ and $c \in \mathrm{IC}_E$, let a truth assignment a_{vc} to the variable x_{vc} be 1 if $c/v \in \sigma$; 0 otherwise. By the supposition, σ contains the binding either w_1/F or w_2/F for each $F \in \mathrm{FV}_E$. Suppose that $w_j/F \in \sigma$ $(j = 1, 2)$. Since $E_F\{w_j/F\}$ is of the form $\{\langle t_{i,j}, s_i\rangle \mid i \in I\}$, it holds that $t_{i,j}\sigma = s_i$. If $t_{i,j} \in \mathrm{IC}_E$, then it holds that $T^F_{i,j} = \mathtt{true}$, since $t_{i,j}\sigma = t_{i,j} = s_i$. Then, $T^F_{i,j}$ is always satisfiable. If $t_{i,j} = v_i \in \mathrm{IV}_E$, then it holds that $s_i/v_i \in \sigma$, since $t_{i,j}\sigma = s_i$. Since $T^F_{i,j}$ is of the form $x_{v_i s_i} \wedge (\bigwedge_{c \in \mathrm{IC} - \{s_i\}} \overline{x_{v_i c}})$, the truth assignment $\{a_{v_i c} \mid c \in \mathrm{IC}_E\}$ satisfies $T^F_{i,j}$. By the definition of T_{E_F}, the truth assignment $a_F = \{a_{v_i c} \mid c \in \mathrm{IC}_E, i \in I\}$ satisfies T_{E_F}, so it satisfies C_{E_F}. Hence, by collecting the truth assignment a_F for each $F \in \mathrm{FV}_E$, C_E is satisfiable.

In the above example, if a matcher σ of E contains the bindings w_1/F and w_2/G, then σ must also contain the bindings a/v, b/y and c/z. For this σ, the satisfiable truth assignment of C_E is constructed such that x_{va}, x_{yb} and x_{zc} are assigned 1 and the other variables 0. □

Concerned with Theorem 3.4, if we replace the condition *predicate* with *ground*, then the following theorem holds.

THEOREM 3.5: BINARYFLATGROUNDMATCHING *is NP-complete. In other words,* MATCHING *is NP-complete even if*

1. *each function variable is at most binary and*
2. *neither function constant nor individual variable occurs.*

Proof: Let C be an instance of MONOTONE 1-IN-3 3SAT over X. Consider the following binary flat ground term language L:

$$L = (\{0,1\}, \emptyset, \emptyset, \{F_{x_1}, G_{x_1}, H_{x_1}, \ldots, F_{x_n}, G_{x_n}, H_{x_n}\}).$$

Here, F_{x_i}, G_{x_i} and H_{x_i} $(1 \leq i \leq n)$ are binary function variables. For each clause $c = z_1 \vee z_2 \vee z_3 \in C$, let E_c be the following matching expression:

$$E_c = \left\{ \begin{array}{ll} \langle F_{z_1}(G_{z_1}(H_{z_2}(0), H_{z_3}(0)), H_{z_1}(0)), 1\rangle, & \\ \langle F_{z_1}(H_{z_1}(1), H_{z_2}(1)), 0\rangle, & \langle G_{z_1}(0,0), 0\rangle, \\ \langle F_{z_2}(G_{z_2}(H_{z_3}(0), H_{z_1}(0)), H_{z_2}(0)), 1\rangle, & \\ \langle F_{z_2}(H_{z_2}(1), H_{z_3}(1)), 0\rangle, & \langle G_{z_2}(0,0), 0\rangle, \\ \langle F_{z_3}(G_{z_3}(H_{z_1}(0), H_{z_2}(0)), H_{z_3}(0)), 1\rangle, & \\ \langle F_{z_3}(H_{z_3}(1), H_{z_1}(1)), 0\rangle, & \langle G_{z_3}(0,0), 0\rangle \end{array} \right\}.$$

Suppose that c is satisfiable by a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ for exactly one index i $(1 \leq i \leq 3)$ and $a_l = 0$ $(1 \leq l \leq 3, l \neq i)$. Then, we can construct a matcher σ of E_c such that $1/H_{z_i}, w_2/F_{z_i} \in \sigma$ if $a_i = 1$ and $0/H_{z_l}, w_1/F_{z_l} \in \sigma$ if $a_l = 0$.

Conversely, suppose that E_c is matchable and σ is a matcher of E_c. Then, σ must contain the binding either w_1/F_{z_i} or w_2/F_{z_i} for each i $(1 \leq i \leq 3)$.

1. If σ contains the bindings w_1/F_{z_1}, w_1/F_{z_2} and w_1/F_{z_3}, then $E_c\sigma$ must be of the following form:

$$E_c\sigma = \left\{ \begin{array}{lll} \langle G_{z_1}(H_{z_2}(0), H_{z_3}(0)), 1\rangle, & \langle H_{z_1}(1), 0\rangle, & \langle G_{z_1}(0,0), 0\rangle, \\ \langle G_{z_2}(H_{z_3}(0), H_{z_1}(0)), 1\rangle, & \langle H_{z_2}(1), 0\rangle, & \langle G_{z_2}(0,0), 0\rangle, \\ \langle G_{z_3}(H_{z_1}(0), H_{z_2}(0)), 1\rangle, & \langle H_{z_3}(1), 0\rangle, & \langle G_{z_3}(0,0), 0\rangle \end{array} \right\}.$$

   In this case, σ must contain the binding either w_1/G_{z_i} or w_2/G_{z_i} for each i $(1 \leq i \leq 3)$. Thus, for some j $(1 \leq j \leq 3)$, $E_c\sigma$ contains both $\langle H_{z_j}(0), 1\rangle$ and $\langle H_{z_j}(1), 0\rangle$, so σ is not a matcher of E_c.

2. If σ contains the bindings w_2/F_{z_1}, w_2/F_{z_2} and w_2/F_{z_3}, then $E_c\sigma$ must be of the following form:

$$E_c\sigma = \left\{ \begin{array}{lll} \langle H_{z_1}(0), 1\rangle, & \langle H_{z_2}(1), 0\rangle, & \langle G_{z_1}(0,0), 0\rangle, \\ \langle H_{z_2}(0), 1\rangle, & \langle H_{z_3}(1), 0\rangle, & \langle G_{z_2}(0,0), 0\rangle, \\ \langle H_{z_3}(0), 1\rangle, & \langle H_{z_1}(1), 0\rangle, & \langle G_{z_3}(0,0), 0\rangle \end{array} \right\}.$$

   By focusing on H_{z_i}, σ is not a matcher of E_c.

3. If σ contains the bindings w_1/F_{z_i}, w_2/F_{z_j} and w_2/F_{z_k} for mutually distinct i, j and k $(1 \leq i, j, k \leq 3)$, then $E_c\sigma$ must be of the following form:

$$E_c\sigma = \left\{ \begin{array}{lll} \langle G_{z_i}(H_{z_j}(0), H_{z_k}(0)), 1\rangle, & \langle H_{z_i}(1), 0\rangle, & \langle G_{z_i}(0,0), 0\rangle, \\ \langle H_{z_j}(0), 1\rangle, & \langle H_{z_k}(1), 0\rangle, & \langle G_{z_j}(0,0), 0\rangle, \\ \langle H_{z_k}(0), 1\rangle, & \langle H_{z_i}(1), 0\rangle, & \langle G_{z_k}(0,0), 0\rangle \end{array} \right\}.$$

   By focusing on H_{z_k}, σ is not a matcher of E_c.

4. If σ contains the bindings w_2/F_{z_i}, w_1/F_{z_j} and w_1/F_{z_k} for mutually distinct i, j and k $(1 \leq i, j, k \leq 3)$, then $E_c\sigma$ must be of the following form:

$$E_c\sigma = \left\{ \begin{array}{lll} \langle H_{z_i}(0), 1\rangle, & \langle H_{z_j}(1), 0\rangle, & \langle G_{z_i}(0,0), 0\rangle, \\ \langle G_{z_j}(H_{z_k}(0), H_{z_i}(0)), 1\rangle, & \langle H_{z_j}(1), 0\rangle, & \langle G_{z_j}(0,0), 0\rangle, \\ \langle G_{z_k}(H_{z_i}(0), H_{z_j}(0)), 1\rangle, & \langle H_{z_k}(1), 0\rangle, & \langle G_{z_k}(0,0), 0\rangle \end{array} \right\}.$$

   Furthermore, σ must contain the bindings w_2/G_{z_j} and w_1/G_{z_k}, and $E_c\sigma$ must be of the following form:

$$E_c\sigma = \left\{ \begin{array}{lll} \langle H_{z_i}(0), 1\rangle, & \langle H_{z_j}(1), 0\rangle, & \langle G_{z_i}(0,0), 0\rangle, \\ \langle H_{z_i}(0), 1\rangle, & \langle H_{z_j}(1), 0\rangle, & \langle 0, 0\rangle, \\ \langle H_{z_i}(0), 1\rangle, & \langle H_{z_k}(1), 0\rangle, & \langle 0, 0\rangle \end{array} \right\}.$$

   In this case, there exists exactly one index i $(1 \leq i \leq 3)$ such that $1/H_{z_i} \in \sigma$ and $0/H_{z_l} \in \sigma$ $(l \neq i)$. Thus we can construct a truth assignment (a_1, a_2, a_3) to (z_1, z_2, z_3) such that $a_i = 1$ if $1/H_{z_i} \in \sigma$ and $a_l = 0$ if $0/H_{z_l} \in \sigma$.

Hence, c is satisfiable by a truth assignment that makes exactly one literal true if and only if E_c is matchable.

For $C = \{c_1, \ldots, c_m\}$, let E be the matching expression $\bigcup_{j=1}^{m} E_{c_j}$ in L. Then, C is satisfiable by a truth assignment that makes exactly one literal of each clause in C true if and only if E is matchable in L. □

Theorem 3.6: UnaryFlatMatching *is solvable in polynomial time. In other words,* Matching *is solvable in polynomial time if*

1. *each function variable is unary and*
2. *no function constant occurs.*

Proof: Let L be a unary flat term language and E be a matching expression in L. For the transformation rule, we adopt the constraint that a projection on F is applied to E if there exist pairs $\langle t_1, s_1\rangle, \langle t_2, s_2\rangle \in E_F$ such that $s_1 \neq s_2$. Since L is unary, the transformation rule can be applied deterministically to E. This algorithm runs in time $O(|E|^2)$. □

3.2. The bounded number of function variables

In this section, we investigate the computational complexity of the restricted problems for kFVMatching.

Theorem 3.7: 1FVTwicePredMatching *is NP-complete. In other words,* Matching *is NP-complete even if*

1. *just one function variable occurs at most twice and*
2. *one function variable does not occur below another function variable.*

Proof: We show this statement by the similar proof to Theorem 3.1.

Let C be an instance of MONOTONE 1-IN-3 3SAT over X. Consider the following 1-fv term language:

$$L = (\{0,1\}, X \cup \{y_1, \ldots, y_m\}, \{f_1, \ldots, f_m, f\}, \{F\}).$$

Here, F and f are m-ary function variable and constant, respectively, and f_j $(1 \leq j \leq m)$ is a binary function constant. For each clause $c_j = x_1^j \vee x_2^j \vee x_3^j \in C$ $(1 \leq j \leq m)$, let s_j, t_j and u_j be the following terms:

$$\begin{array}{rcl} s_j & = & f_j(x_3^j, f_j(x_2^j, f_j(x_1^j, y_j))), \\ t_j & = & f_j(0, f_j(0, f_j(1, f_j(0,0)))), \\ u_j & = & f_j(0, f_j(0,0)). \end{array}$$

Then, let E be the following read-twice predicate matching expression in L:

$$E = \{\langle F(s_1, \ldots, s_m), f(t_1, \ldots, t_m)\rangle,\ \langle F(y_1, \ldots, y_m), f(u_1, \ldots, u_m)\rangle\}.$$

Suppose that C is satisfiable by the truth assignment $a = (a_1, \ldots, a_n)$ that makes exactly one literal of each clause in C true. For each $c_j \in C$, let (a_1^j, a_2^j, a_3^j) denote the truth assignment to x_1^j, x_2^j and x_3^j from a. Then, there exists exactly one index i_j $(1 \leq i_j \leq 3)$ such that $a_{i_j}^j = 1$ and $a_l^j = 0$ $(1 \leq l \leq 3, l \neq i_j)$ for each j $(1 \leq j \leq m)$. Let $\rho(l,n) = ((l+n-2) \bmod 3) + 1$ $(1 \leq l, n \leq 3)$. Consider the following L^*-terms p_j and q_j for each j.

1. If $i_j = 1$ $(a_1^j = 1)$, then $p_j = w_j$ and $q_j = f_j(0, f_j(0,0))$.
2. If $i_j = 2$ $(a_2^j = 1)$, then $p_j = f_j(0, w_j)$ and $q_j = f_j(0,0)$.
3. If $i_j = 3$ $(a_3^j = 1)$, then $p_j = f_j(0, f_j(0, w_j))$ and $q_j = 0$.

Hence, the following substitution σ is a matcher of E.

$$\sigma = \{f(p_1, \ldots, p_m)/F\} \cup \{1/x_{\rho(i_j,1)}, 0/x_{\rho(i_j,2)}, 0/x_{\rho(i_j,3)}, q_j/y_j \mid 1 \leq j \leq m\}.$$

Conversely, suppose that E is matchable in L. By Corollary 2.1 and by the form of E, E is matchable in L if and only if so is the matching expression E' in the term language L' as follows:

$$\begin{array}{rcl} E' & = & \{\langle H_j(s_1, \ldots, s_m), t_j\rangle,\ \langle H_j(y_1, \ldots, y_m), u_j\rangle \mid 1 \leq j \leq m\}, \\ L' & = & (\mathrm{IC}_L, \mathrm{IV}_L, \mathrm{FC}_L, \mathrm{FV}_L \cup \{H_1, \ldots, H_m\}). \end{array}$$

Let σ be a matcher of E'. By the forms of s_j, t_j and u_j, σ contains one of the following bindings for each j $(1 \leq j \leq m)$.

1. w_j/H_j, $1/x_1^j$, $0/x_2^j$, $0/x_3^j$ and $f_j(0, f_j(0,0))/y_j$,
2. $f_j(0, w_j)/H_j$, $0/x_1^j$, $1/x_2^j$, $0/x_3^j$ and $f_j(0,0)/y_j$, or
3. $f_j(0, f_j(0, w_j))/H_j$, $0/x_1^j$, $0/x_2^j$, $1/x_3^j$ and $0/y_j$.

Since E' is matchable, it is uniquely determined whether each $x_i \in X$ is substituted to 0 or 1. By using the bindings with x_i in the substitution, we can construct the truth assignment to X that makes exactly one literal of each clause in C true. □

On the other hand, if we add the condition *flat* to kFVMatching, then the following theorem holds.

Theorem 3.8: kFVFlatMatching *is solvable in polynomial time for each* $k \geq 0$. *In other words,* Matching *is solvable in polynomial time if*

1. *at most* k *different function variables occur and*
2. *no function constant occurs.*

Proof: Let L be a k-fv flat term language with k function variables $F_1, \ldots, F_k$ and E be a matching expression in L. Let n be the maximum arity of F_i $(1 \leq i \leq k)$. We adopt the same constraint of Theorem 3.6. Since L is flat, E contains no function constants, so when we once apply an imitation or a projection to E it decreases at least one function variable in E. Furthermore, a projection is applied to E at most n times for every function variable. Hence, we can determine whether E is matchable or not by checking at most n^k first-order matching expressions. Here, this algorithm runs in time $O(|E|^k)$. □

3.3. Predicate matching expressions in ground term languages

Theorem 3.3 and 3.5 claim that both FlatPredMatching and FlatGroundMatching are NP-complete in general. On the other hand, the following theorem holds.

Theorem 3.9: GroundPredMatching *is solvable in polynomial time. In other words,* Matching *is solvable in polynomial time if*

1. *no individual variable occurs and*
2. *no function variable occurs below other function variables.*

Proof: Let L be a ground term language and E be a predicate matching expression in L. Consider the following two projections, instead of a projection:

1. **projection 1** (on F): if there exists an index i such that E_F is of the form $\{\langle F(t_1^1, \ldots, t_n^1), t_i^1\rangle, \ldots, \langle F(t_1^m, \ldots, t_n^m), t_i^m\rangle\}$, then
$$E \;\Rightarrow\; E\{w_i/F\}.$$
2. **projection 2** (on F): if E_F does not satisfy the above condition and there exist pairs $\langle F(t_1, \ldots, t_n), s_1\rangle, \langle F(u_1, \ldots, u_n), s_2\rangle \in E_F$ such that $\mathrm{hd}(s_1) \neq \mathrm{hd}(s_2)$, then
$$E \;\Rightarrow\; \texttt{fail}.$$

An imitation on F is applied to E if E does not satisfy the above conditions in projections 1 and 2 on F. Then, the transformation rule $\Rightarrow$ is applied deterministically to E.

Since L is ground and E is predicate, E is transformed to `fail` by a projection 2 if and only if $E \not\Rightarrow^* \emptyset$ by only an imitation and a simplification. Furthermore, by an imitation on F and a simplification, the right-hand term of pairs in E_F is decomposed into the proper subterms. Hence, by Theorem 2.1, the statement holds, where this algorithm runs in time $O(|E|^2)$. □

4. The Comparison between Second-Order Matching and Unification Problems

In this section, we compare the separations of tractable second-order matching problems from intractable ones with the separations of decidable second-order unification problems from undecidable ones.

1. Amiot [1] (and implicitly Farmer [10]) has shown that the unification problem is undecidable for read-twice predicate unification expressions in unary term languages with at least one binary function constant. On the other hand, by Theorem 3.1, the matching problem is NP-complete for read-twice predicate matching expressions in unary term languages with at least one binary function constant.

2. Farmer has shown that the unification problem is decidable in monadic term languages [9], but undecidable in nonmonadic unary term languages with at least one binary function constant [10]. On the other hand, by Theorem 3.1 and 3.2, both of the corresponding matching problems are NP-complete.

3. Goldfarb [14] has shown that the unification problem is undecidable in ternary ground term languages. On the other hand, by Theorem 3.2, the matching problem is NP-complete in unary ground term languages. Note that Amiot's and Farmer's results [1, 10] do not imply that the unification problem is undecidable in unary ground term languages, because the existence of individual variables is essential in their proofs.

4. As pointed by Goldfarb [14], the unification problem is decidable in flat term languages. On the other hand, by Theorem 3.3 or 3.5, the matching problem is NP-complete in flat term languages. However, it is solvable in polynomial time for predicate matching expressions in binary flat term languages, in unary flat term languages, or in k-fv ($k \geq 0$) flat term languages by Theorem 3.4, 3.6 or 3.8.

5. Ganzinger *et al.* [13] have shown that the unification problem is undecidable for read-twice predicate matching expressions in 1-fv term languages. On the other hand, by Theorem 3.7, the matching problem is NP-complete for read-twice predicate matching expressions in 1-fv term languages.

6. Levy and Veanes [23] have shown that the unification problem is undecidable

in 1-fv ground or unary term languages. Whether the corresponding matching problems are NP-complete is still open.

7. Schubert [26] has shown that the unification problem is undecidable for predicate unification expressions in ground term languages. On the other hand, by Theorem 3.9, the matching problem is solvable in polynomial time for predicate matching expressions in ground term languages.

8. Dowek [8] has shown that the unification problem is decidable for read-once unification expressions, and the matching problem is solvable in linear time for read-once matching expressions. Furthermore, Levy [22] has shown that the unification problem is undecidable for read-twice predicate unification expressions. On the other hand, Theorem 3.1 or 3.7 claims that the matching problem is NP-complete for read-twice predicate matching expressions even if term languages are either unary or 1-fv.

5. Conclusion

We summarize the results obtained by this paper as Table 1.

term language	expression	matching	reference	unification	reference
–	–	NP-complete	[2]	undecidable	[14]
Unary	TwicePred	NP-complete	Theorem 3.1	undecidable	[1, 10]
TernaryGround	–	NP-complete	Theorem 3.2	undecidable	[14]
UnaryGround	–	NP-complete	Theorem 3.2		
TernaryFlat	ThricePred	NP-complete	Theorem 3.3	decidable	[14]
BinaryFlat	Pred	poly time	Theorem 3.4	decidable	[14]
BinaryFlatGround	–	NP-complete	Theorem 3.5	decidable	[14]
UnaryFlat	–	poly time	Theorem 3.6	decidable	[14]
1FV	TwicePred	NP-complete	Theorem 3.7	undecidable	[13]
1FVGround	–	open		undecidable	[23]
1FVUnary	–	open		undecidable	[23]
kFVFlat ($k \geq 0$)	–	poly time	Theorem 3.8	decidable	[14]
Ground	Pred	poly time	Theorem 3.9	undecidable	[26]
Mon	–	NP-complete	Theorem 3.2	decidable	[9]
NonmonUnary	–	NP-complete	Theorem 3.1	undecidable	[10]
–	Once	linear time	[8]	decidable	[8]
–	TwicePred	NP-complete	Theorem 3.1,3.7	undecidable	[22]

Table 1: The complexity of second-order matching and unification problems

The existence of function constants and individual variables works essentially for separating tractable second-order matching problems from intractable ones and decidable second-order unification problems from undecidable ones. In particular, the nonexistence of function constants makes the second-order unification problems decidable [14], but does not separate tractable second-order matching problems from intractable ones. On the other hand, the nonexistence of individual variables makes the second-order matching problems tractable for the predicate expressions, but does not separate decidable second-order unification problems from undecidable ones.

In this paper, we have dealt with the second-order matching problems for matching expressions consisting of L-terms, but not one consisting of L^*-terms. The later derives the matching problem of *second-order patterns* [24, 25], which is related to the problem GroundPredMatching: If we can regard the bound variables in L^*-terms as individual constants, then it is related to the problem GroundPredMatching. Furthermore, the second-order matching for matching expressions consisting of L^*-terms is also related to the *pure* matching problem: Wierzbick [28] has shown that the second- and third-order pure matching is NP-complete and the fourth-order one is NEXPTIME-hard.

Curien *et al.* [4] have designed a complete second-order matching algorithm which works more efficient than the one of Huet [17] and Huet and Lang [18] in most cases. Roughly speaking, their algorithm is regarded as the algorithm of GroundPredMatching with the recursive calls for the nested function variables. When it is necessary to obtain the complete set of matchers for a given matching expression, we know no more efficient algorithm than their algorithms although it is not a polynomial-time algorithm. From the viewpoint of applications, on the other hand, there are many cases necessary to extract the optimum matcher in some sense rather than to extract all of the matchers. We are investigating it [20, 29, 30] in the framework of *schema matching* and *analogical reasoning*. It is a future work to give the trade-off between completeness and efficiency of the second-order matching adequate for each research field.

Acknowledgment

The authors would like to thank anonymous referees for valuable comments to revise the preliminary version of this paper.

References

[1] G. Amiot, *The undecidability of the second order predicate unification problem*, Archive for Mathematical Logic **30**, 193–199, 1990.

[2] L. D. Baxter, *The complexity of unification*, Doctoral Thesis, Department of Computer Science, University of Waterloo, 1977.

[3] H. Comon, Y. Jurski, *Higher-order matching and tree automata*, Selected Papers of 11th International Workshop on Computer Science Logic, LNCS **1414**, 157–176, 1997.

[4] R. Curien, Z. Qian, H. Shi, *Efficient second-order matching*, Proc. 7th International Conference on Rewriting Techniques and Applications, LNCS **1103**, 317–331, 1996.

[5] G. de Moor, G. Sittampalam, *Generic program transformation*, Lect. 3rd International School on Advanced Functional Programming, LNCS **1608**, 116–149, 1998.

[6] G. Défourneaux, C. Bourely, N. Peltier, *Semantic generalizations for proving and disproving conjectures by analogy*, Journal of Automated Reasoning **20**, 27–45, 1998.

[7] M. R. Donat, L. R. Wallen, *Learning and applying generalized solutions using higher order resolution*, Proc. 9th International Conference on Automated Deduction, LNCS **310**, 41–60, 1988.

[8] G. Dowek, *A unification algorithm for second-order linear terms*, manuscript, 1993. Also available at `http://coq.inria.fr/~dowek/`.

[9] W. M. Farmer, *A unification algorithm for second-order monadic terms*, Annals of Pure and Applied Logic **39**, 131–174, 1988.

[10] W. M. Farmer, *Simple second-order languages for which unification is undecidable*, Theoretical Computer Science **87**, 25–41, 1991.

[11] P. Flener, *Logic program synthesis from incomplete information*, Kluwer Academic Press, 1995.

[12] M. R. Garey, D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, 1979.

[13] H. Ganzinger, F. Jacquemard, M. Veanes, *Rigid reachability*, Proc. 4th Asian Computing Science Conference, LNCS **1538**, 4–11, 1998.

[14] W. D. Goldfarb, *The undecidability of the second-order unification problem*, Theoretical Computer Science **13**, 225–230, 1981.

[15] M. Harao, *Proof discovery in LK system by analogy*, Proc. 3rd Asian Computing Science Conference, LNCS **1345**, 197–211, 1997.

[16] G. P. Huet, *A unification algorithm for typed λ-calculus*, Theoretical Computer Science **1**, 27–57, 1975.

[17] G. P. Huet, *Résolution d'equations dans les languages d'ordre* $1, 2, \ldots, \omega$ Thèse d'Etat, Universitè de Paris VII, 1976.

[18] G. P. Huet, B. Lang, *Proving and applying program transformations expressed with second-order patterns*, Acta Informatica **11**, 31–55, 1978.

[19] T. Kolbe, C. Walther, *Proof analysis, generalization and reuse*, W. Bibel, P. H. Schmitt (eds.), *Automated deduction – A basis for applications*, Vol. II, Chapter 8, Kluwer Academic Publishers, 189–219, 1998.

[20] K. Kubo, K. Yamada, K. Hirata, M. Harao, *Efficient schema matching algorithm based on pre-checking*, Transactions of IEICE **J85-D-I**, 143–151, 2002 (in Japanese).

[21] J. Levy, *Linear second-order unification*, Proc. 7th International Conference on Rewriting Techniques and Applications, LNCS **1103**, 332–346, 1996.

[22] J. Levy, *Decidable and undecidable second-order unification problem*, Proc. 9th International Conference on Rewriting Techniques and Applications, LNCS **1379**, 47–60, 1998.

[23] J. Levy, M. Veanes, *On the undecidability of second-order unification*, Information and Computation **159**, 125–150, 2000.

[24] D. Miller, *A logic programming language with lambda-abstraction, function variables, and simple unification*, Journal of Logic and Computation **1**, 497–536, 1991.

[25] C. Prehofer, *Decidable higher-order unification problems*, Proc. 12th International Conference on Automated Deduction, LNAI **814**, 635–649, 1994.

[26] A. Schubert, *Second-order unification and type inference for Church-style polymorphism*, Proc. 25th ACM Symposium on Principle of Programming Languages, 279–288, 1998.

[27] W. Snyder, J. Gallier, *Higher-order unification revisited: Complete sets of transformations*, Journal of Symbolic Computation **8**, 101–140, 1989.

[28] T. Wierzbicki, *Complexity of the higher order matching*, Proc. 16th International Conference on Automated Deduction, LNAI **1632**, 82–96, 1999.

[29] K. Yamada, K. Hirata, M. Harao, *Schema matching and its complexity*, Transactions of IEICE **J82-D-I**, 1307–1316, 1999 (in Japanese).

[30] K. Yamada, K. Hirata, M. Harao, *Second-order schema matching based on projection point labeling*, Proc. 15th International Workshop on Unification, Technical Report DII 09/01, Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Siena, 49–53, 2001.