Set-Term Matching in a Logic Database Language

Seung Jin Lim
Computer Science Department
Brigham Young University
Provo, Utah 84602, U.S.A.
Email: sjlim@dragon.cs.byu.edu

Yiu-Kai Ng
Computer Science Department
Brigham Young University
Provo, Utah 84602, U.S.A.
Email: ng@cs.byu.edu

Abstract

Existing set-term matching algorithms [1] for logic-based database languages, of which set terms have the commutative and idempotent properties, have several problems: Given a pair of set terms, matchers can only be computed sequentially and duplicated matchers are generated. Hence, these algorithms cannot take the advantage of existing multiple processors for computing all matchers in parallel. Further, duplicated matchers are redundant and undesirable. In order to overcome these shortcomings, we propose an improved set-term matching approach for LDL/NR, a logic database language for nested relations, which generates non-redundant matchers for a given pair of set terms in parallel. The approach thus solves the sequential and redundant problems in [1].

1 Introduction

Logic-based data languages, such as LDL [6], HILOG [4], HILOG-R [5], LDL/PNF [2], and LDL/NR (Logic Database Language for Nested Relations) [3], extend logic programming from first-order logic to high-order logic to handle (recursive) database queries on complex data structures (i.e., non-first-normal-form relations). A common data structure used in these languages is set terms which are of the form \{e_1, \ldots, e_n\}, where each \(e_i\), \(1 \leq i \leq n\), is distinct and may be complex in structure, and a key design issue of these languages is unification, which is a computation tool widely used in automated deduction [10, 11].

In this paper, we investigate the problem of set-term matching which is considered as one-way set-term unification since set-term matching is commonly used in deductive database systems for handling bottom-up computation. We assume that each element of a set term \(S\) is either a constant or a variable as in [1]. Further, \(S\) is treated as the concatenation of a number of elements, i.e., as an ordered list. However, since a set term represents a set and since ordering and repetition of elements are immaterial in a set, the concatenation operation is enhanced with the properties of commutativity and idempotency [1, 6].

Existing set-term matching approaches for generating matchers of two given set terms [1, 6] in a logic database language either are inefficient or compute matchers sequentially with duplicates. These set-term matching problems must be solved in order to process queries in LDL/NR. LDL/NR is of interest because it captures the constraints of nested relations precisely. Further, since for each complex-object type there is a nested-relation type with the same “information capacity” [7], LDL/NR can handle complex data-type queries. In this paper, we propose two different set-term matching approaches for LDL/NR. One of these approaches generates matchers in parallel, and both approaches generate matchers without duplicates.

2 Basic Definitions

2.1 LDL/NR

LDL/NR is based on the notions of type, term, formula, rule, fact, and query, which in turn are defined on an alphabet [3]. Constants and variables in LDL/NR are of atomic type. There are two other types: tuple type and set type, defined in LDL/NR. Set type and tuple type can be used alternatively to form complex data types, as in [4, 5].

Definition 1 A type is recursively defined as follows:
(i) an atomic type is a type, (ii) a set type \(S\{r\}\) is a type, where \(r\) is either an atomic type or a tuple type, and (iii) a tuple type \(p(s_1, \ldots, s_n)\) is a type, where each \(s_i\), \(1 \leq i \leq n\), is either an atomic type or a set type. 

From now on whenever we refer set-terms, we mean set-terms with the commutative and idempotent properties unless stated otherwise.
Example 1 The declaration dept(Dname, projects {Pname}, employees { employee (Ename, EID)) defines dept which is of tuple type with components Dname, projects, and employees. Dname is of atomic type, and projects, which includes all projects controlled by department Dname, and employees, which includes all employees who work for department Dname, are of set types.

Definition 2 A term is recursively defined as follows: (i) a constant is a term, (ii) a variable is a term, (iii) for a set-type s[r], s{t1, . . . , tm} is a term called set term, where each ti, 1 ≤ i ≤ m, is of type r, and (iv) for a tuple-type p(s1, . . . , sn), p(t1, . . . , tn) is a term called tuple term, where ti is of type si, 1 ≤ i ≤ n.

Example 2 An instance of the tuple-type dept is dept(cs, projects{db, se, Zp}, employees {employee (smith, 123), employee(jones, 567)}).

2.2 Set Terms in LDL/NR

In this subsection, we define set-term matchers and the commutative and idempotent properties of set terms in LDL/NR. The commutative property of a set term S in LDL/NR allows elements in S to be arranged in any order, while the idempotent property of S treats duplicated elements in S as a single element.

Definition 3 A set term S in LDL/NR satisfies the commutative property if given any two elements e1 and e2, concat(e1, concat(e2, S)) = concat(e2, concat(e1, S)). A set term S in LDL/NR satisfies the idempotent property if given any element e, concat(e, concat(e, S)) = concat(e, S).

As in other logic database languages for complex data structures, in LDL/NR the assignment of a term to a variable is called a binding, and a finite set of bindings is called a substitution.

Definition 4 Let S1 and S2 be two set terms in LDL/NR. A substitution θ is a unifier for S1 and S2 if S1 o θ = S2 o θ. If either S1 or S2 is a ground term, then θ is a matcher for S1 and S2.

Example 3 Let S1 = {a, b} and S2 = {X, Y, b} be two set terms in LDL/NR. Then, θ1 = {X/a, Y/a}, θ2 = {X/a, Y/b}, and θ3 = {X/b, Y/a} are all matchers of S1 and S2. Thus, there can be more than one matcher for any given pair of set terms in LDL/NR.

3 Set-Term Matching Approaches for LDL/NR

In this section, we present the approaches for set-term matching adopted by our algorithms in [8]. The first approach is sequential but simple, while the other one is parallel. It is assumed that given any two set terms S1 and S2, S1 and S2 satisfy the following condition, called the matching condition, as in [1].

Definition 5 (Matching Condition) Given any two set terms S1 and S2 in LDL/NR, S1 and S2 are of the following form:

\[
S_1 = \{a_1, \ldots, a_m, a_{m+1}, \ldots, a_{m+n}\} \\
S_2 = \{X_1, \ldots, X_m, \ldots, X_{m+p}, a_{m+1}, \ldots, a_{m+n}\}
\]

where ai (1 ≤ i ≤ m + n) denotes a constant, Xi (1 ≤ i ≤ m + p) denotes a variable, a1, . . . , am are called m-compliants, and am+1, . . . , am+n are common constants. It is further required that there exist at least one constant in S1 and at least one variable in S2. Hence, m + n > 0, m + p > 0, and m, n, p ≥ 0.

Hereafter, whenever we refer two set terms S1 and S2, S1 and S2 are assumed to satisfy the matching condition unless stated otherwise.

Definition 6 If the cardinality of a set, subset, multiset, or permutation Δ is l, then Δ is called an l-set, l-subset, l-multiset, or l-permutation, respectively.

Definition 7 Given S1 and S2, let κ = {a1, . . . , am} be the set of m-compliants in S1 and let θ = {V1/t1, . . . , Vn/tn} be a substitution. A subset (multiset) Δ of S1 is an m-idempotent subset (multiset) or we say that Δ satisfies m-idempotency if \{a1, . . . , am\} ⊆ Δ. If κ ⊆ {t1, . . . , tn}, then θ satisfies m-idempotency.

Example 4 Let S1 = {a, b, c} and S2 = {X, Y, Z, c} be two set terms in LDL/NR. Then, {a, b} is an 2-idempotent 2-subset of S1, {a, a, b} is an 2-idempotent 3-multiset of S1, and θ = {X/a, Y/a, Z/b} satisfies m-idempotency.

Proposition 1 Given two set terms S1 and S2, let M = {m1, . . . , mm+p} be a multiset, where mi (1 ≤ i ≤ m + p) is a constant in S1. A substitution σ = \{X1/m1, . . . , Xm+p/mm+p\} constructed by M and all the variables in S2 is a matcher of S1 and S2 if and only if M satisfies m-idempotency.

Proof. (If) Let M = {m1, . . . , mm+p} be an (m + p)-multiset that satisfies m-idempotency. Then, by Definition 7, M must include all m-compliants a1, . . . , am. Further, the set of variables in S2 is \{X1, . . . , Xm+p\}. Hence, given M and S2, we can construct the substitution σ = \{X1/m1, . . . , Xm+p/mm+p\}, and
$S_2 \circ \sigma = S_1$. By Definition 4, $\sigma$ is a matcher of $S_1$ and $S_2$.

(Only if) Since $\sigma = \{X_1/m_1, \ldots, X_{m+p}/m_{m+p}\}$ is a matcher of $S_1$ and $S_2$, $S_2 \circ \sigma = S_1$. Then, $S_2 \circ \sigma - \{a_{m+1}, \ldots, a_{m+p}\} = \{a_1, \ldots, a_m\}$. Hence, $\{a_1, \ldots, a_m\} \subseteq M$, and $M$ satisfies $m$-idempotency. □

Example 5 Let $S_1 = \{a, b\}$ and $S_2 = \{X, Y, b\}$. A 2-multiset $M = \{a, b\}$ of $S_1$ satisfies 1-idempotency, and $\sigma = \{X/a, Y/b\}$ is a matcher of $S_1$ and $S_2$ since $S_2 \circ \sigma = S_1$. □

Proposition 2 Given two set terms $S_1$ and $S_2$, let $P = \{p_1, \ldots, p_{m+p}\}$ be a permutation that is generated from an $m$-idempotent $(m+p)$-multiset $M$. Then, the substitution $\{X_1/p_1, \ldots, X_{m+p}/p_{m+p}\}$ constructed by using $P$ and all the variables in $S_2$ is a matcher of $S_1$ and $S_2$.

Proof. Any permutation of an $m$-idempotent $(m+p)$-multiset $M$ satisfies $m$-idempotency. By Proposition 1, the substitution $\{X_1/p_1, \ldots, X_{m+p}/p_{m+p}\}$ constructed by using $P$ and all the variables in $S_2$ is a matcher of $S_1$ and $S_2$. □

Example 6 Let $S_1, S_2$, and $M$ be as given in Example 5. $\{b, a\}$ is a permutation of $M$ which can be used for generating the substitution $\sigma = \{X/b, Y/a\}$. Since $S_2 \circ \sigma = S_1$, $\sigma$ is a matcher of $S_1$ and $S_2$. □

3.1 A Simple Approach for Solving the Set-Term Matching Problem in LDL/NR

In this subsection, we propose a simple approach for solving the set-term matching problem in LDL/NR which generates all distinct matchers of two given set terms $S_1$ and $S_2$. These matchers are generated by computing all distinct $m$-idempotent $(m+p)$-permutations of $S_1$ (as shown in Proposition 2). We compute these permutations by using an improved permutation algorithm $PA$ which constructs all distinct $m$-idempotent $(m+p)$-permutations of $S_1$ by ordered selections of the constants in $S_1$ in the lexicographical order\(^2\) of the indices of the constants in $S_1$. $PA$ thus enhances the permutation algorithms in [1] since the latter adopts a classical permutation algorithm which assumes that an input consists of distinct elements and hence generates duplicated permutations for computed multisets as a result. Our simple approach constructs distinct $(m+p)$-permutations and eliminates all non-$m$-idempotent $(m+p)$-permutations at the same time.

3.1.1 Computing All Distinct $(m+p)$-permutations

The following strategies are adopted to compute all distinct $(m+p)$-permutations: (i) From left to right, assign each constant in $S_1$ an index $i$ which represents the $i$th constant in $S_1$. (ii) From left to right, assign each component of an $(m+p)$-permutation a constant in $S_1$ which is chosen according to the lexicographical order of the indices of constants in $S_1$. (iii) A constant in $S_1$ can be chosen up to $m+p$ times as distinct components of an $(m+p)$-permutation.

Proposition 3 There are $N_{all} = (m+n)^{m+p}$ distinct $(m+p)$-permutations of constants in $S_1$.

Proof. Since there are $m+n$ distinct constants in $S_1$ and each of these constants can be chosen up to $m+p$ times, there are $(m+n)^{m+p}$ distinct $(m+p)$-permutations of constants in $S_1$. □

3.1.2 Eliminating Non-$m$-idempotent $(m+p)$-permutations

While generating distinct $(m+p)$-permutations of $S_1$, we eliminate every partial $(m+p)$-permutation\(^3\) which violates the pre-$m$-idempotent condition as defined below. If a partial permutation $P$ violates the pre-$m$-idempotent condition, then a fully constructed $(m+p)$-permutation $P'$ generated from $P$ does not satisfy $m$-idempotency. Hence, whenever a partial permutation $P$ violates the pre-$m$-idempotent condition, the construction process of $P$ is terminated immediately.

Definition 8 (Pre-$m$-idempotent condition) A partial $(m+p)$-permutation $P$ satisfies the pre-$m$-idempotent condition if whenever a non-$m$-compliant in $S_1$ is chosen as a component of $P$, the number of unbound variables in $S_2$ is greater than the number of unassigned $m$-compliants.

Proposition 4 The number of all possible non-$m$-idempotent $(m+p)$-permutations of $S_1$ is

$$N_m = \sum_{i=1}^{m} (-1)^i \times \binom{m}{i} \times (m+n-i)^{m+p}.$$  

Proof. A non-$m$-idempotent $(m+p)$-permutation $P$ has less than $m$ $m$-compliants which means that $i$ compliants are excluded from $P$. Let $P_i, 1 \leq i \leq m$, denote all $(m+p)$-permutations without $i$ compliants. Since there are $(m+n-i)^{m+p}$ possible permutations in $P_i$, $|P_1 \cup \ldots \cup P_m| = N_m$. □

\(^2\)During the construction process of an $m$-idempotent $(m+p)$-permutation $P$, constants in $S_1$ are chosen as components of $P$ according to the index values, instead of the actual values, of the constants.

\(^3\)A partial $(m+p)$-permutation $P$ is a permutation being constructed such that there exists at least one component of $P$ that has not been assigned a value. If every component of $P$ is assigned a value, $P$ is called a fully constructed (or simply a) permutation.
3.1.3 Generating All m-idempotent (m + p)-permutations

Based on the discussion above, we now describe how to generate each m-idempotent (m+p)-permutation \( P' = \{e_1, \ldots, e_{m+p}\} \) from \( S_1 = \{a_1, \ldots, a_{m+n}\} \). Let \( P_i (1 \leq i \leq m+p) \) denote a partial \((m+p)\)-permutation. We assume that each component \( e_1, \ldots, e_{i-1} \) in \( P_i \) has already been assigned a constant in \( S_1 \). In order to generate all m-idempotent (m+p)-permutations of \( S_1 \), we construct an \((m+n)\)-ary tree \( T \) of height \( m + p \), called the permutation tree. Each node \( N \) at level \( i \) (1 ≤ i ≤ m+p) in \( T \) denotes a different \( P_i \). Each component \( e_i \) in each \( P_i \) at level \( i \) is assigned a distinct constant in \( S_1 \).

A leaf node in \( T \) denotes a \( P' \) in \( S_1 \). A constant \( C \) in \( S_1 \) is usable for \( P_i \) if assigning \( C \) as the \( i \)-th component of \( P_i \) does not cause \( P_i \) to violate the pre-m-idempotent condition.

\( T \) is constructed as follows: (i) Construct the root node of \( T \). (ii) For each node \( N \) at level \( i \) (0 ≤ i < m+p - 1), create \( k \) (1 ≤ k ≤ m+n) children of \( N \) such that the partial \((m+p)\)-permutation \( P_{i+1} \) which is associated with a child of \( N \) does not violate the pre-m-idempotent condition. For \( P_{i+1} \) which is associated with the \( j \)-th child of \( N \) (1 ≤ j ≤ k), \( e_{i+1} \) in \( P_{i+1} \) is the \( j \)-th usable constant in \( S_1 \). (iii) Terminate the construction of \( T \) when each m-idempotent \((m+p)\)-permutation, which is associated with a leaf node of \( T \), has been generated.

Example 7 Figure 1 illustrates the entire construction process of the \( m \)-ary permutation tree \( T \) of height \( 3 \) which yields all 2-idempotent \( 3 \)-permutations of \( S_1 = \{a,b,c\} \), given \( S_2 = \{X,Y,Z,c\} \).

3.2 A Parallel Approach for Solving the Set-Term Matching Problem in LDL/NR

In the simple approach, permutations are generated directly from two given set terms \( S_1 \) and \( S_2 \) in a sequential manner. In this section, we present a parallel approach for solving the set-term matching problem in LDL/NR. Algorithm 2 in [8] is the direct implementation of the parallel approach. Prior to discussing the parallel approach, we give the definitions of subset group and multiset group.

Definition 9 Let \( S \) be a set term, \( S' = \{\Delta_1, \ldots, \Delta_n\} \) is a subset group of \( S \) if \( 0 \leq l \leq |S|, |\Delta_l| \subseteq S \), and \( |\Delta_l| = l, 1 \leq i \leq n \). \( G^l = \{M_1, \ldots, M_k\} \) is a multiset group of \( S \), where \( l \) denotes the number of distinct constants in each \( m \)-idempotent \((m+p)\)-multiset \( M_i^l \) (1 ≤ i ≤ k).

Example 8 Given \( S = \{a, b, c\} \), all the possible subset groups of \( S \) are \( S^1 = \{\{a\}, \{b\}, \{c\}\} \), \( S^2 = \{\{a,b\}, \{a,c\}, \{b,c\}\} \), and \( S^3 = \{\{a,b,c\}\} \). If 'a' is the only compliant, then all the possible 1-idempotent \( S \)-multiset groups of \( S \) are \( G^1 = \{\{a, a, a\}\} \), \( G^2 = \{\{a, a, b\}, \{a, b, b\}, \{a, a, c\}, \{a, c, c\}\} \), and \( G^3 = \{\{a, b, c\}\} \).

In order to compute all the matchers of \( S_1 \) and \( S_2 \) in parallel, we (i) generate all \( m \)-idempotent disjoint subsets in each subset group of \( S_1 \), (ii) derive all \( m \)-idempotent \((m+p)\)-multiset groups from the subset groups generated in (i), (iii) determine all permutations of each multiset in each multiset group, and (iv) apply variable substitution over the variables in \( S_2 \) to the components of each permutation generated in (iii) which yields a matcher of \( S_1 \) and \( S_2 \).

3.2.1 Generating Subset Groups in Parallel

Given two set terms \( S_1 \) and \( S_2 \), we determine in parallel all subset groups \( S^m, S^{m+1}, \ldots, S^{\min(m+n,m+p)} \) of \( S_1 \) by choosing \( m \) and \( \min(m+n,m+p) \) constants from \( S_1 \). These subset groups include all \( m \)-idempotent subsets of \( S_1 \), are disjoint, and are computed in parallel. The arguments follow:

1. To obtain all \( m \)-idempotent subsets in each subset group, we exclude each subset group \( S^l \), \( 0 \leq l \leq m-1 \) or \( l > \min(m+n,m+p) \). \( S^l \), \( 0 \leq l \leq m-1 \), is excluded since there does not exist any \( m \)-idempotent subset in \( S^l \) with less than \( m \) elements. \( S^l \), \( l > \min(m+n,m+p) \), is excluded since it is impossible to choose more than \( m+n \) distinct constants from \( S_1 \) to construct \( S^l \). Further, \( S^l \), \( l > \min(m+n,m+p) = m+n \), is
is excluded since we cannot include each chosen constant in an \((m+p)\)-subset of \(S_1\) to generate an \((m+p)\)-multiset of \(S_1\) if we have to choose more than \(m+p\) constants from \(S_1\).

2. Each subset group \(S'_l\), \(m \leq l \leq \min(m+n,m+p)\), is disjoint since for any \(\Delta'_l \in S'_l\) and \(\Delta'_{l+1} \in S'_{l+1}\), \(|\Delta'_l| \neq |\Delta'_{l+1}|\). Further, subsets in each subset group \(S'_l\) are disjoint since they are generated according to the lexicographical order of indices of the elements in \(S_1\).

3. Since choosing \(i\) elements from \(S_1\) to form \(S'_l\) and choosing \(j\) elements from \(S_2\) to form \(S'_j\), \(i \neq j\), can be done independently, each subset group can be generated in parallel.

Before introducing the approach for generating all \(m\)-idempotent subset groups, we need the definition of left-leaning-tree which is used during the construction process of each subset group.

**Definition 10** An \(n\)-ary left-leaning tree \(T\) of height \(h\) satisfies the following constraints: (i) The root node has \(n\) children. (ii) A node \(p\) at level \(i\) in \(T\) has \(k\) children, \(1 \leq i \leq h-1\), \(1 \leq k \leq n\), and the \(j\)th child of \(p\) has \(m\) children, \(1 \leq j \leq k\), \(m = k-j+1\). (iii) At level \(i\), \(1 \leq i \leq h\), there are \(\binom{n+i-1}{i}\) nodes.

The entire process of generating each subset group \(S'_l\), \(m \leq l \leq \min(m+n,m+p)\), in parallel can be done by constructing a forest \(F\), which is a collection of \((m+n-l+1)\)-ary left-leaning trees \(T_i\) of height \(l\). Each \(T_i\) in \(F\) represents the process of generating \(S'_l\) and each node \(N\) at level \(i\) (\(1 \leq i \leq l\)) in \(T_i\) denotes a different partial \(l\)-subset \(SB = \{e_1, \ldots, e_i\}\) in \(S'_l\). We assume that at the parent node of \(N\), each component \(e_1, \ldots, e_{i-1}\) in \(SB\) has already been assigned a constant in \(S_1\). A leaf node in \(T_i\) denotes a fully constructed \(l\)-subset \(SB'\).

We construct \(F\) as follows: (i) Construct the root node of each \(T_i\) in \(F\), \(m \leq l \leq \min(m+n,m+p)\), in parallel. (ii) For each node \(N\) at level \(i\) (\(0 \leq i \leq l-1\)) in \(T_i\), create \(k\) (\(1 \leq k \leq m+n-l+1\)) children of \(N\). At a node \(M\) at level \(j\) (\(1 \leq j \leq l\)) in \(T_i\), \(e_i\) in \(SB\), which is associated with node \(M\), is set to the \((k_1+k_2)\)-th constant in \(S_1\), where \(k_{i-1}\) is the \(k_1\)-th constant in \(S_1\), \(SB\) is the \(k_2\)-th child of its parent, and \(k_1 = 0\) when \(i = 1\). (iii) Terminate the construction of \(T_i\) when all the leaf nodes of \(T_i\) have been constructed.

Leaf nodes in \(T_i\) yield all the subsets \(SB'\) in \(S'_l\); however, only those subsets \(\Delta'\) which satisfy \(m\)-idempotency are used for computing \((m+p)\)-multisets of \(S_1\). From now on, whenever we refer to a subset group \(S'_l\), it is assumed that \(S'_l\) consists of \(m\)-idempotent subsets only.

**Example 9** Figure 2 illustrates the process of generating all \((m\)-idempotent and non-\(m\)-idempotent\) subsets in the subset group \(S^3\) with \(S_1 = \{a, b, c, d, e\}\). In the figure, \(X\) at a node denotes an unbound component of a subset being constructed.

**Example 10** Given \(S_r = \{u, b, c\}\) and \(S_2 = \{x, y, z, b\}\), \(m = 2\), \(n = 1\) and \(p = 1\). Since \(m = 2\) and \(\min(m+n,m+p) = 3\), we generate subset groups \(S_2 = \{\{a, c\}\}\) and \(S^3 = \{\{a, b, c\}\}\), and exclude all non-\(m\)-idempotent subsets of \(S_1\). Subsets \(\{a, b\}\) and \(\{b, c\}\) are excluded from \(S^3\) since they do not satisfy \(m\)-idempotency.

**Proposition 5** Given two set terms \(S_1\) and \(S_2\), \(|S'_l| = \binom{n}{i}\), where \(m \leq l \leq \min(m+n,m+p)\) and \(i = l - m\), and for each subset \(\Delta'_l \in S'_l\), \(\Delta'_l\) satisfies \(m\)-idempotency.

**Proof.** Since no subset in \(S'_l\), \(1 \leq m\) or \(l > \min(m+n,m+p)\), satisfies \(m\)-idempotency as discussed earlier, only subset groups \(S'_l\) (\(m \leq l \leq \min(m+n,m+p)\)) yields \(m\)-idempotent subsets. Further, since we can choose \(l - m\) constants from \(\{a_{m+1}, \ldots, a_{m+n}\}\) in addition to the compliants \(a_1, \ldots, a_m\) to form \(m\)-idempotent subsets, there are \(\binom{n}{l - m}\) \(m \leq l \leq \min(m+n,m+p), i = l - m\), as shown in the second column of Table 1.

3.2.2 Generating Multiset Groups in Parallel

We generate each \(m\)-idempotent \((m+p)\)-multiset group \(G'_l\) from each \(m\)-idempotent subset \(\Delta'_l\) in subset group \(S'_l\), \(m \leq l \leq \min(m+n,m+p)\), by appending constants chosen from \(\Delta'_l\) in \(S'_l\) to \(\Delta'_l\), if necessary. Since we construct each \(M'_l\) in \(G'_l\) by using all the constants in \(\Delta'_l\) which satisfies \(m\)-idempotency, \(M'_l\) also satisfies \(m\)-idempotency. Further, by adopting the same strategy used for generating subset groups, we
Table 1: Number of m-idempotent subsets and m-idempotent (m + p)-multisets, where \( K = \min(m + n, m + p) \).

<table>
<thead>
<tr>
<th>Subset group</th>
<th>Number of m-idempotent subsets in a subset group</th>
<th>Number of m-idempotent (m + p)-multisets from a subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g^1 )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( g^{m-1} )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( g^m )</td>
<td>(m + p - 1)</td>
<td>(m + p - 1)</td>
</tr>
<tr>
<td>( g^{m+1} )</td>
<td>(m + p - 1)</td>
<td>(m + p - 1)</td>
</tr>
<tr>
<td>( g^{m+2} )</td>
<td>(m + p - 1)</td>
<td>(m + p - 1)</td>
</tr>
<tr>
<td>( g^k )</td>
<td>( \min(n, p) )</td>
<td>( \min(n, p) )</td>
</tr>
<tr>
<td>Total</td>
<td>( \sum_{i=0}^{\min(n, p)} (m + p - i) )</td>
<td>( \sum_{i=0}^{\min(n, p)} (m + p - i) )</td>
</tr>
</tbody>
</table>

Example 11 Consider Example 10 again where \( \Delta_1^2 = \{a, c\} \) and \( \Delta_2^2 = \{a, b, c\} \). We can construct multisets \( G^2 \) and \( G^3 \), the two 2-idempotent 3-multisets computed from \( \Delta_1^2 \) and \( \Delta_2^2 \), respectively, where \( G^2 = \{M_1^2 = \{a, c, c\}, M_2^2 = \{a, c, c\}\} \) and \( G^3 = \{M_1^3 = \{a, b, c\}\} \). □

Proposition 6 Given two set terms \( S_1 \) and \( S_2 \), \( |G^l| = \left( \begin{array}{c} n \\ i \end{array} \right) \times \left( \begin{array}{c} m + p - 1 \\ p - i \end{array} \right) \), where \( m \leq i \leq \min(m + n, m + p) \) and \( i = l - m \), i.e., \( 0 \leq i \leq \min(n, p) \).

Proof. Since each m-idempotent subset group \( S^l \) consists of a number of subsets and each subset \( \Delta_1^l \) in \( S^l \) generates \( N \) m-idempotent (m + p)-multisets, \( |G^l| = |S^l| \times N \), where \( |S^l| = \left( \begin{array}{c} n \\ i \end{array} \right) \) as shown in Proposition 5. Further, given \( i \) distinct constants \( \{b_1, \ldots, b_i\} \) in \( \Delta_1^l \), \( m \leq i \leq \min(m + n, m + p) \), we can assign each of these constants at least once (up to \( m + p - 1 \) times) over \( m + p \) components of an (m + p)-multiset. The number of these assignments is indeed the number of non-negative integer solutions of the equation \( x_1 + \ldots + x_{l-1} = m + p - 1 \), \( x_j \geq 0 \), where \( x_j \) denotes the number of \( b_j \), \( 1 \leq j \leq l \), in \( \Delta_1^l \) that has been chosen. Thus, the number of these assignments is \( \left( \begin{array}{c} m + p - 1 \\ l - 1 \end{array} \right) = \left( \begin{array}{c} m + p - 1 \\ p - i \end{array} \right) \). Hence, the number of multisets \( N \) generated from a given subset \( \Delta_1^l \) is \( \left( \begin{array}{c} m + p - 1 \\ p - i \end{array} \right) \), \( 0 \leq i \leq \min(n, p) \) (and \( i = l - m \)), as shown in the third column of Table 1, and the number of m-idempotent (m + p)-multisets generated from an \( l \)-subset group \( S^l \) is \( |G^l| \). □

Proposition 7 Given \( S_1 \) and \( S_2 \), the number of m-idempotent (m + p)-multisets of \( S_1 \) is \( \sum_{i=0}^{\min(n, p)} \left( \begin{array}{c} n \\ i \end{array} \right) \times \left( \begin{array}{c} m + p - 1 \\ p - i \end{array} \right) \).

Proof. By Proposition 6, a subset group \( S^{m+i} \), \( 0 \leq i \leq \min(n, p) \), generates \( \left( \begin{array}{c} n \\ i \end{array} \right) \times \left( \begin{array}{c} m + p - 1 \\ p - i \end{array} \right) \) multisets. Hence, the total number of m-idempotent (m + p)-multisets is the summation over the subset groups \( S^m \) to \( S^{\min(m + n, m + p)} \) which yields the above formula. □

3.2.3 Generating Permutations in Parallel

The permutation algorithm used in the parallel approach accepts an m-idempotent (m + p)-multiset \( M^l \), \( m \leq i \leq \min(m + n, m + p) \), as input and generates all distinct permutations of \( M^l \) as output. An inventory vector \( \text{inv} = \{v_1, \ldots, v_l\} \) is created to keep track of the number of each distinct constant in \( M^l \) that can be used for generating a permutation of \( M^l \), and each \( v_i, 1 \leq i \leq l \), in \( \text{inv} \) denotes the number of \( i \)-th distinct constant in \( M^l \), and \( \sum_{i=1}^{2} v_i = m + p \).
Permutations of \( M' \) are computed by choosing constants in \( M' \) in the lexicographical order of the indices of the constants. Assume that \( \text{inv} \) has been created. Then, each permutation \( P' \) of \( M' = \{ e_1, \ldots, e_{m+p} \} \) can be generated by constructing an \( i \)-ary tree \( T \) of height \( m + p \). Each node \( N \) at level \( i \) (\( 1 \leq i \leq m + p \)) in \( T \) denotes a different partial \((m + p)\)-permutation \( P = \{ p_1, \ldots, p_{m+p} \} \). If \( 1 \leq j \leq m + p \), in each \( P \) at level \( i \) is assigned a different constant in \( M' \). A leaf node in \( T \) denotes a fully constructed \( P' \).

We construct \( T \) as follows: (i) Construct the root node of \( T \). (ii) For each node \( N \) at level \( i \), \( 0 \leq i \leq m + p - 1 \), create \( k \) (\( 1 \leq k \leq I \)) children of \( N \), where \( k \) is the number of non-zero components of \( \text{inv} \). (\( 1 \leq j \leq m + p \)) in each partial permutation \( P \), which is associated with a node \( R \) at level \( j \) (\( 1 \leq j \leq m + p \)), is set to the \( k \)th distinct usable constant in \( M' \), and \( R \) is the \( k \)th child of its parent. Decrement \( v_k \) in \( \text{inv} \) by 1 after the \( k \)th distinct constant in \( M' \) has been chosen. (iii) Terminate the construction of \( T \) when each permutation of an \( m \)-idempotent \((m + p)\)-multiset, which is associated with a leaf node of \( T \), has been generated.

Example 12 Consider \( M_f \) in Example 11 again. The initial inventory vector \( \text{V} \) of \( M_f \) is \( <2, 1> \) which denotes that there are two 'a's and one 'c', and every possible permutation of \( M_f \) is generated in the lexicographical order of the indices of 'a' and 'c' (i.e., 1 for 'a' and 2 for 'c'). The process of generating all permutations for \( M_f = \{ a, c, u \} \) is illustrated in Figure 4, where \( X \) stands for a component in a partial permutation that has not been assigned any value.

3.3 Complexity Analysis of the Proposed Approaches

In this paper we present only the approaches of the set-term matching algorithms and not the detailed algorithms. Hence, the proofs included in the following subsections are only verifications of the approaches; however, the verifications are in fact also proofs of the algorithms in [6].

3.3.1 Complexity Analysis of the Simple Approach

Proposition 8 The number of \( m \)-idempotent \((m + p)\)-permutations generated by the simple approach is

\[
\sum_{i=0}^{m} \left(-1\right)^i \times \binom{m}{i} \times (m + n - i)^{m+p}
\]

Proof. The number of \( m \)-idempotent \((m + p)\)-permutations \( N \) generated by the simple approach can be computed by subtracting the number of non-\( m \)-idempotent \((m + p)\)-permutations from the number of all possible \((m + p)\)-permutations. Thus,

\[
N = N_{all} - N_{loss} = \left( m + n \right)^{m+p} - \sum_{i=1}^{m} \left(-1\right)^i \times \binom{m}{i} \times (m + n - i)^{m+p}
\]

which is simplified as Equation 1, where \( N_{all} \) and \( N_{loss} \) are as computed in Proposition 3 and Proposition 4, respectively.

Theorem 1 The time complexity of the simple approach is

\[
O \left( \sum_{i=0}^{m} \left(-1\right)^i \times \binom{m}{i} \times (m + n - i)^{m+p} \right)
\]

Proof. Given each \( m \)-idempotent \((m + p)\)-permutation, the simple approach performs a variable substitution for the \( m + p \) variables in \( S_{2} \) to generate a matcher of \( S_{1} \) and \( S_{2} \). Thus, each permutation requires additional \( m + p \) unit time for variable substitution. Hence, the time complexity of the simple approach is as shown in the equation above.

3.3.2 Complexity Analysis of the Parallel Approach

Proposition 9 \( MAX_{sb-sp} \), the number of subsets in a subset group generated from a given set, is \( \sqrt{\frac{2}{n}} \times 2^n \).

Proof. For each subset group \( S_i \), the parallel approach generates \( \binom{n}{i} \) subsets, where \( 0 \leq i \leq \min(n, p) \) and \( i = l - m \), which follows from the first half of the proof for Proposition 6. Further, for any given \( n \), if \( \min(n, p) = n \), then the largest binomial coefficient exists at \( \frac{n}{2} \) according to the binomial theorem [9]. Also, if \( \frac{n}{2} \leq p < n \), then the largest coefficient also appears at \( \frac{n}{2} \), and if \( p < \frac{n}{2} \), then \( \left( \binom{n}{j} \right) < \left( \binom{n}{2} \right) \), \( 0 \leq j \leq p \). Therefore, \( MAX_{sb-sp} = \left( \binom{n}{2} \right) = \frac{n!}{(n/2)!^2} \times \frac{2^n}{2^n} \).

By the Stirling's formula [12],

\[
\frac{n!}{(n/2)!^2} \approx \left( \frac{2}{\pi} \right)^{n/2} \frac{1}{\sqrt{2\pi n}}
\]

\[
\sqrt{\frac{2}{n}} \times 2^n.
\]

```
Proposition 10 \( \text{MAX}_{mset} \), the number of \( m \)-idempotent \((m + p)\)-multisets generated from a subset by the parallel approach, is \( \sqrt{\frac{2}{(m+p-1)!}} \times 2^{m+p-1} \).

Proof. For each subset \( S_i \), the parallel approach generates \( \binom{m+p-1}{p-i} \) multisets, where \( 0 \leq i \leq \min(n, p) \) and \( i = l-m \), which follows from the second half of the proof for Proposition 6. By the binomial theorem, \( \text{MAX}_{mset} = \binom{m+p-1}{m+p-1} \) and, applying the Stirling’s formula [12] to \( \text{MAX}_{mset} \) yields the desired result. \( \Box \)

Proposition 11 To generate a permutation \( P \) from an \((m+p)\)-multiset, the number of components of \( P \) to be assigned is \( \text{MAX}_{perm} = m + p \).

Proof. Since given any \((m+p)\)-multiset \( M \), there are \( m + p \) components to be assigned in parallel to each permutation of \( M \), \( \text{MAX}_{perm} = m + p \). \( \Box \)

Theorem 2 The time complexity of the parallel approach is \( \mathcal{O} \left( (m + p) \times \left( n^{\frac{1}{2}} \times 2^n \right) \times \left( (m+p)^{\frac{1}{2}} \times 2^{m+p} \right) \right) \).

Proof. The parallel approach generates subsets, multisets, and permutations accordingly. Since each permutation requires the \( m+p \) unit time to generate a matcher by variable substitution for the \( m+p \) variables, the time complexity of the parallel approach is \( \mathcal{O} \left( (m + p) \times (\text{MAX}_{sb-gp} \times \text{MAX}_{mset} + \text{MAX}_{perm}) \right) \) which is the formula given above. \( \Box \)

3.4 Proofs of Correctness of the Set-Term Matching Approaches

Theorem 3 The simple approach is sound and complete.

Proof. The simple approach generates all possible permutations of \((m + p)\)-multisets of \( S_1 \) as shown in Proposition 3, and eliminate all non-\( m \)-idempotent permutations as shown in Proposition 4. By Proposition 2, the simple approach generates all \( m \)-idempotent \((m + p)\)-multisets which, after variable substitution, yield all matchers of \( S_1 \) and \( S_2 \). \( \Box \)

Theorem 4 The parallel approach is sound and complete.

Proof. The parallel approach generates all \( m \)-idempotent \((m + p)\)-multisets of \( S_1 \) and \( S_2 \) as shown in Proposition 6 and Proposition 7. These multisets are permuted to generate all possible permutations. These permutations are converted into matchers by variable substitutions. By Proposition 2, the parallel approach generates all the matchers of \( S_1 \) and \( S_2 \). \( \Box \)

4 Summary and Future Works

We have presented set-term matching approaches for LDL/NR that generate non-redundant matchers for a given pair of set terms and enhance the algorithms in [1] by computing matchers in parallel. In the future, we want to study the set-term unification problem in LDL/NR and provide set-term unification algorithms that generate no-redundant unifiers in parallel.

References


