Unification of Multisets with Multiple Labelled Multiset Variables *

Zan Naeem 1 and Giselle Reis 1

Carnegie Mellon University, Doha, Qatar
znaeem@andrew.cmu.edu, giselle@cmu.edu

Abstract

We look into the problem of unifying multisets containing (first-order) terms and multiple multiset variables. The variables are labelled, meaning that a unifier that places a term in a multiset variable $M_i$ is different from another that places a term in a multiset variable $M_j$, for $i \neq j$. We describe a sound, complete, and terminating algorithm for computing the set of all possible unifiers, and analyse its complexity. We also prove an input pre-processing step that avoids the computation of less general unifiers.

1 Introduction

Multiset is an important data-structure that is used to specify various object systems. Our motivation stems mostly from proof theory, where logical entailment is encoded as sequents $\Gamma \vdash \Delta$, where both $\Gamma$ and $\Delta$ are typically considered as multisets. When reasoning about such objects, one might need to use an implementation of sets-multisets based on lists, since most reasoning tools (i.e., logical frameworks, proof assistants, and logic programming languages) do not have built-in support for these data structures [2, 3, 6, 7]. Adding this kind of support requires, among other things, a unification algorithm.

Multiset unification was studied in [1, 4], where the authors propose solutions for the problem of unifying multisets with at most one multiset variable. We extend those results for multisets with multiple multiset variables. In our setting, each multiset variable is labelled, meaning that assigning a term to either a multiset $M_i$ or $M_j$, where $i \neq j$, should be considered different solutions. We describe a terminating algorithm and analyse its complexity. Moreover, we prove that a simple modification of our algorithm avoids the computation of less general unifiers.

The need for labelled multiset variables emerged when reasoning with multiplicative rules in sequent calculi, such as:

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \rightarrow B \vdash C}$$

To apply this rule to, e.g., the sequent $\Gamma, D, A \rightarrow B \vdash C$, where $A, B, C$, and $D$ are formulas, we need to unify its antecedent $\Gamma, D, A \rightarrow B$ with $\Gamma_1, \Gamma_2, A \rightarrow B$. Assigning formula $D$ to $\Gamma_1$ or $\Gamma_2$ should be considered two different solutions, since they result in two different applications of the rule.

1.1 Preliminaries

A multiset $\mathcal{M}$ with multiple labelled multiset variables is denoted by $\{t_1, ..., t_n | M_1, ..., M_k\}$. Each $t_i$ is a term ranging over a first-order language $\mathcal{L} = \langle \Sigma, V \rangle$, where $\Sigma$ is a set of constants

---

*This publication was made possible by the support of Qatar Foundation through Carnegie Mellon University in Qatar’s Seed Research program. The statements made herein are solely the responsibility of the authors.
and function symbols, and \( \mathcal{V} \) is a denumerable set of term variables. Each \( M_i \) is a multiset variable ranging over a denumerable set \( \mathcal{V}_M \) of multiset variables. When \( n \) and \( k \) are not relevant, we abbreviate \( t_1, \ldots, t_n \) as \( \overline{t} \) and \( M_1, \ldots, M_k \) as \( \overline{M} \). When \( k = 0 \), we write the multiset as \( \{t_1, \ldots, t_n\} \). Henceforth, we refer to multisets with multiple labelled multiset variables as mmsets for brevity.

**Definition 1** (Substitution). An mmset substitution \( \sigma \) is a finite mapping of term variables to terms, and of multiset variables to mmsets. The application of a substitution \( \sigma \) to an mmset \( \{t_1, \ldots, t_n|M_1, \ldots, M_k\} \) is defined as:

\[
\{t_1, \ldots, t_n|M_1, \ldots, M_k\} \sigma = \{t_1\sigma, \ldots, t_n\sigma\} \uplus M_1\sigma \uplus \ldots \uplus M_k\sigma
\]

where each \( t_i\sigma \) is the usual first-order substitution and \( \uplus \) is left-associative and defined as:

\[
\{t_1, \ldots, t_n|M_1, \ldots, M_k\} \uplus \{s_1, \ldots, s_m|N_1, \ldots, N_l\} = \{t_1, \ldots, t_n, s_1, \ldots, s_m|M_1, \ldots, M_k, N_1, \ldots, N_l\}
\]

**Definition 2** (Equality). Mmsets \( M = \{t_1, \ldots, t_n|M_1, \ldots, M_k\} \) and \( N = \{s_1, \ldots, s_m|N_1, \ldots, N_l\} \) are considered equal modulo a constraint theory \( T \), written \( M =_T N \) iff: \( n = m \) and \( t_1, \ldots, t_n \) is a permutation of \( s_1, \ldots, s_m \); and \( T \vdash M_1 \cup \ldots \cup M_k \equiv N_1 \cup \ldots \cup N_l \).

## 2 Mmsets Unification

The mmset unification problem of mmsets \( M_1 \) and \( M_2 \) consists of finding a substitution \( \sigma \) and constraint theory \( T_\sigma \) such that \( M_1 \sigma =_{T_\sigma} M_2 \sigma \). The theory \( T_\sigma \) consists of an equality over unions of multiset variables, and it is computed a posteriori for each unifier \( \sigma \).

### 2.1 Algorithm

In what follows, we use \( \sigma \) to denote a single substitution, \( \Sigma \)s to denote sets of substitutions, \( \times \) for the Cartesian product of two sets (or lists), and \( \setminus \) for multiset difference. The pseudo code for all algorithms are listed in Appendix A, and an implementation in SML can be found at https://github.com/meta-logic/mmset-unif.

The main function for mmset unification is implemented by Algorithm 1. In the most general case (lines 11 to 17), the unifiers of \( \{\overline{t}|\overline{M}\} \) and \( \{\overline{s}|\overline{N}\} \) are computed by choosing a subset of terms (of the same size) from \( \overline{t} \) and \( \overline{s} \) to be unified, and distributing the rest among the multiset variables \( \overline{M} \) and \( \overline{N} \). The number of terms chosen to be unified can vary from 0 to the minimum length of \( \overline{t} \) and \( \overline{s} \). Two other cases are considered separately for efficiency purposes. The first one is when there are no multiset variables (line 1). Here a unification is only possible if \( |\overline{t}| = |\overline{s}| \). The second case is when one of the mmsets does not have multiset variables (lines 4 and 7). If \( \overline{M} \) is empty, then all terms in \( \overline{s} \) must be unified with a term from \( \overline{t} \). The remaining terms in \( \overline{t} \) can be allocated in \( \overline{N} \).

Function \texttt{unify}\_\texttt{c} (Algorithm 2) chooses \( c \) terms from the multisets \( \overline{t} \) and \( \overline{s} \) to be unified, and distributes the rest of the terms among the multiset variables. The function \texttt{choose}(\( F, c \)) returns a set of tuples \( (F_c, F_c) \), where \( F_c \) is the multiset with the chosen \( c \) elements (thus \( |F_c| = c \)), and \( F_c = F \setminus F_c \). Unifiers for the chosen terms are computed by \texttt{unify}\_\texttt{terms} and stored in \( \Sigma_t \). Substitutions for each context variable, containing the remaining terms, are computed by \texttt{unify}\_\texttt{distribute} and stored in \( \Sigma_M \). The final set of unifiers consists of the composition \( \sigma_M \cdot \sigma_t \) for each \( (\sigma_M, \sigma_t) \in \Sigma_M \times \Sigma_t \). Note that, since the image of \( \sigma_M \) may contain terms, the composition needs to be in this order.
Function `unify_terms` (Algorithm 3) finds all unifiers of two multisets of terms (without multiset variables) of equal length. This is done by testing all possible pairings of the two multisets, obtained by pairing some order of the first multiset with all possible permutations of the second one. For each pairing, the function `unify_lists` computes the most general unifier.

Function `unify_distribute` (Algorithm 4) computes unifiers for the mmsets \{t[M]\} and \{τ[N]\} considering that all t occurs in N and all τ occurs in M. Let \( \Sigma_N \) denote the substitutions that distribute t into N, and \( \Sigma_M \) the substitutions that distribute \( \tau \) into M. The resulting substitutions are \( \sigma_M \cup \sigma_N \) for each \( (\sigma_M, \sigma_N) \in \Sigma_M \times \Sigma_N \). A simple union can be used in the case, as the image of \( \sigma_M \) is disjoint from the domain of \( \sigma_N \) (see below).

Function `distribute` (Algorithm 5) is used by `unify_distribute` and that is where the aforementioned \( \Sigma_N \) and \( \Sigma_M \) are computed. It computes substitutions for the multiset variables \( N_1, ..., N_l \) such that all terms \( t_1, ..., t_n \) occur in one of the multisets. This is done by calculating all ordered \( l \)-partitions of the multiset \( \{t_1, ..., t_n\} \). For example, the ordered 2-partitions of the multiset \( \{a, b, c\} \) are:

\[
\begin{align*}
\{\{a, b\}, \{c\}\} & \quad \{\{a, b\}, \{c\}\} & \quad \{\{a\}, \{b, c\}\} & \quad \{\{a\}, \{b\}\} & \quad \{\{a\}, \{b, c\}\} & \quad \{\{b\}, \{a, c\}\} & \quad \{\{b\}, \{a, c\}\}
\end{align*}
\]

Each computed substitution corresponds to an \( l \)-ordered partition. If there are no terms \((n = 0)\), then there is only the trivial partition of \( l \) empty multisets. In this case, the algorithm returns a list with one substitution, which maps every multiset variable \( N_i \) to the mmset with no terms. If there are no multiset variables \((l = 0)\), then there are no partitions, and thus no possible substitutions. The exceptional case is when there are no terms nor multiset variables \((l = n = 0)\). In this case, the solution is the set containing only the empty substitution (\{\}).

The last parameter of `distribute` indicates whether the multiset variables should contain exactly the terms \( t_1, ..., t_n \). If set to true, then there is no more space for other terms, and \( N_i \) is mapped to an mmset with the appropriate terms and no multiset variables. Otherwise it is mapped to an mmset with a set of terms and a fresh multiset variable. If there are no terms to place in the multiset variable, it is mapped to itself (to avoid unnecessary renamings). This is needed to compute the constraint theory, after which such identity substitutions can be eliminated.

**Constraint theory** For a unifier \( \sigma \), the constraint theory \( T_\sigma \) is defined as:

\[
\bigcup\{M'_i \mid M_i \mapsto \{ts|M'_i\} \in \sigma\} = \bigcup\{N'_i \mid N_i \mapsto \{ts|N'_i\} \in \sigma\}
\]

Soundness and completeness of the algorithm are straightforward, since it exhaustively checks all possibilities for unifying multisets.

**Theorem 1.** *Soundness* If `unify`\((M,N) \mapsto \{\sigma_1, ..., \sigma_n\}\) then \( \forall \sigma_i \cdot M\sigma_i = T_\sigma \cdot N\sigma_i \)

**Theorem 2.** *Completeness* If \( \exists \sigma \cdot M\sigma = T_\sigma \cdot N\sigma \) then `unify`\((M,N) \mapsto \{\sigma_1, ..., \sigma_n\}\) and \( \exists \sigma_i \) such that \( \sigma = \sigma_i \sigma' \).

Note that the use of a substitution \( \sigma' \) is needed even if the set of computed unifiers is not the minimal one.
2.2 Complexity

The most expensive part of the unification algorithm is the one between lines 11 and 17 in Algorithm 1, so we concentrate our complexity analysis to that case. For each $i$ from 0 to the minimum number of terms, \textsf{unify}_{\mathcal{L}}$ is called. This function has two nested loops over the sets $T_c$ and $S_c$ (Alg. 2, lines 4, 5), which contain all possible ways of choosing $i$ elements from $n$ and $m$, respectively. Thus $|T_c| = \binom{n}{i}$ and $|S_c| = \binom{m}{i}$. In the inner part of the loops, \textsf{unify\_terms} is called, which finds all possible unifiers for two multisets of size $i$. Since all possible pairings of elements must be tried, and for each order \textsf{unify\_lists} runs in $i^2$, the function (Alg. 3) has complexity $i^2!$. The function \textsf{unify\_distribute} (Alg. 4) computes all possible ways of partitioning $n - i$ elements into $l$ parts, and $m - i$ elements into $k$ parts, and returns the Cartesian product of these sets. Therefore, its complexity is $l^{n-i} k^{m-i}$.

Putting those together, we get to the cost for the unification of multisets $\{t_1, \ldots, t_n| M_1, ..., M_k\}$ and $\{s_1, ..., s_m| N_1, ..., N_l\}$:

$$\sum_{i=0}^{\min(n, m)} \binom{n}{i} \binom{m}{i} i^2! l^{n-i} m^{m-i}$$

After some arithmetic manipulation, we can conclude that, on the worst case, \textsf{unify} runs in $O(n! m! l^n k^m)$. For the special case where the multisets have only one multiset variable: $l = k = 1$, and the unification algorithm runs in $O(n! m!)$.

2.3 Removing Less General Unifiers

The algorithm described in Section 2.1 does not compute the set of minimal unifiers. For example, given multiset $\{a, a|M\}$ and $\{a|N\}$, \textsf{unify} computes three unifiers with constraint: $\{N \mapsto \{a|N'\}\}$, where $M \equiv N'$, twice (once for each occurrence of $a$), and $\{M \mapsto \{a|M'\} ; N \mapsto \{a|N'\}\}$, where $M' \equiv N'$. These are the same unifiers obtained by the non-deterministic algorithm from [5].

In order to reduce the number of less general unifiers, we can remove every pair of equal terms $t_i$ and $s_j$ from the multisets (i.e., $t_i$ and $s_j$ unify with the empty substitution). The rationale behind this is that, every other unifier that is obtained by unifying these terms with something else, or placing them in a multiset variable, can be recovered from the set of unifiers obtained when this pair is not in the multiset.

We start by showing that it is safe to eliminate pairs of equal terms from the problem of unifying multisets without multiset variables.

**Theorem 3.** Let $\mathcal{T}$ and $\mathcal{S}$ be two multisets of terms such that $t_a \in \mathcal{T}$ and $s_b \in \mathcal{S}$ are equal, for some $a$ and $b$. Let $\Sigma_{all} = \textsf{unify\_terms}(\mathcal{T}, \mathcal{S})$, and $\Sigma = \textsf{unify\_terms}(\mathcal{T} \setminus \{t_a\}, \mathcal{S} \setminus \{s_b\})$. Then for every $\sigma \in \Sigma_{all}$, there exists $\mu \in \Sigma$ s.t. $\sigma = \mu \sigma'$ for some substitution $\sigma'$.

The proof for this theorem can be found in Appendix B. The overall idea is as follows. $\sigma$ was obtained by some pairing of terms in $\mathcal{T}$ and $\mathcal{S}$. We choose $\mu$ as the unifier that used a pairing that is as close as possible as the one used for $\sigma$. Those pairings differ only for the terms involving $t_a$ and $s_b$. Suppose $t_a$ is paired with $s_x$ and $s_b$ is paired with $t_y$. Using the most general unifiers of $t_y$ and $s_x$, we can conclude the existence of $\sigma'$ such that $\sigma = \mu \sigma'$.

**Theorem 4.** Let $\{t|M\}$ and $\{\pi|N\}$ be two multisets such that $t_a \in \mathcal{T}$ is equal to $s_b \in \mathcal{S}$ for some $a$ and $b$. Moreover, let $\Sigma_{all} = \textsf{unify}(\{t|M\}, \{\pi|N\}$, and $\Sigma = \textsf{unify}(\mathcal{T} \setminus \{t_a\}|\mathcal{M}, \mathcal{S} \setminus \{s_b\}|\mathcal{N})$. Then for every $\sigma \in \Sigma_{all}$ there exists $\mu \in \Sigma$ such that $\sigma = \mu \sigma'$ for some $\sigma'$.
The proof for this theorem can be found in Appendix B. We proceed by a case analysis on whether \( t_a \) and \( s_b \) were chosen to be unified, or to be placed in a multiset variable. There are four cases. The case in which both are chosen to be unified is solved using Theorem 3. For the case in which both are placed in a multiset variable, we can construct \( \sigma' \). The other two (dual) cases are the more involved ones. They use a combination of the two strategies of the first cases.

This modification is implemented in the algorithm available online, and extensive testing has shown that all less general unifiers are eliminated. In particular, only the unifier \( \{N \mapsto \{a|N''\}\} \), where \( M \equiv N' \) is computed for mmsets \( \{a, a|M\} \) and \( \{a|N\} \).

3 Conclusion

We have developed a sound and complete algorithm for finding unifiers of multisets with multiple multiset variables. The algorithm is deterministic and terminating. It is implemented in SML, and we also provide the pseudo code for reproducibility. The same algorithm can be used for the particular case where there is only one multiset variable.

The complexity of the unification procedure is analysed, and its cost is high. This is inherent to the problem, since it is of combinatorial nature. It may be possible to improve this result by using the right data-structures and heuristics, but for our purposes, since the numbers are quite small, it runs fast enough.

We have also tried to eliminate all sources of redundancy, so that the set of computed unifiers is as close as possible to the minimal one. In particular, we have shown that a simple pre-processing of the input problem will produce fewer unifiers, and all those that are no longer produced can be recovered. We conjecture that this optimization leads to the computation of the minimal set of unifiers, but we leave this investigation as future work.

References

A Algorithms

Algorithm 1\[\text{unify}(\{t_1, \ldots, t_n|M_1, \ldots, M_k\}, \{s_1, \ldots, s_m|N_1, \ldots, N_l\})\]
1: \text{if } k = 0 \land l = 0 \text{ then}
2: \text{if } n = m \text{ then return } \text{unify}_\text{terms}([t_1, \ldots, t_n], [s_1, \ldots, s_m])
3: \text{else return } []
4: \text{else if } k = 0 \text{ then}
5: \text{if } m \leq n \text{ then return } \text{unify}_\text{c}(m, \{t_1, \ldots, t_n|M_1, \ldots, M_k\}, \{s_1, \ldots, s_m|N_1, \ldots, N_l\})
6: \text{else return } []
7: \text{else if } l = 0 \text{ then}
8: \text{if } n \leq m \text{ then return } \text{unify}_\text{c}(n, \{t_1, \ldots, t_n|M_1, \ldots, M_k\}, \{s_1, \ldots, s_m|N_1, \ldots, N_l\})
9: \text{else return } []
10: \text{else}
11: c \leftarrow \min(n, m)
12: \Sigma \leftarrow []
13: \text{for } i = 0 \text{ to } c \text{ do}
14: \Sigma' \leftarrow \text{unify}_\text{c}(i, \{t_1, \ldots, t_n|M_1, \ldots, M_k\}, \{s_1, \ldots, s_m|N_1, \ldots, N_l\})
15: \Sigma \leftarrow \Sigma \cup \Sigma'
16: \text{end for}
17: \text{return } \Sigma
18: \text{end if}

Algorithm 2\[\text{unify}_\text{c}(c, \{t_1, \ldots, t_n|M_1, \ldots, M_k\}, \{s_1, \ldots, s_m|N_1, \ldots, N_l\})\]
1: \Sigma \leftarrow []
2: \text{T}_c \leftarrow \text{choose}(\{t_1, \ldots, t_n\}, c)
3: \text{S}_c \leftarrow \text{choose}(\{s_1, \ldots, s_m\}, c)
4: \text{for } (\{t'_1, \ldots, t'_c\}, \{t'_{c+1}, \ldots, t'_n\}) \in \text{T}_c \text{ do}
5: \text{for } (\{s'_1, \ldots, s'_c\}, \{s'_{c+1}, \ldots, s'_m\}) \in \text{S}_c \text{ do}
6: \Sigma_c \leftarrow \text{unify}_\text{terms}([t'_1, \ldots, t'_c], [s'_1, \ldots, s'_c])
7: \text{if } \Sigma_c \neq [] \text{ then}
8: \Sigma_M \leftarrow \text{unify}_\text{distribute}((\{t'_{c+1}, \ldots, t'_n\}, \{M_1, \ldots, M_k\}), (\{s'_{c+1}, \ldots, s'_m\}, \{N_1, \ldots, N_l\}))
9: \Sigma' \leftarrow \text{map}(\lambda (\sigma_t, \sigma_M).\sigma_M \sigma_t)(\Sigma_c \times \Sigma_M)
10: \Sigma \leftarrow \Sigma \cup \Sigma'
11: \text{end if}
12: \text{end for}
13: \text{end for}
14: \text{return } \Sigma
Algorithm 3  
\textit{unify\_terms}([t_1, \ldots, t_n], [s_1, \ldots, s_n])

1. $\Sigma \leftarrow \emptyset$
2. $P_s \leftarrow \text{permutations}([s_1, \ldots, s_n])$
3. $P \leftarrow [[t_1, \ldots, t_n]] \times P_s$
4. for $(T, S) \in P$ do
5. \hspace{1em} $\sigma \leftarrow \text{unify\_lists}(T, S)$
6. \hspace{1em} if $\sigma \neq \text{None}$ then $\Sigma \leftarrow \{\sigma\} \cup \Sigma$
7. end for

Algorithm 4  
\textit{unify\_distribute}((\{t_1, \ldots, t_n\}, \{M_1, \ldots, M_k\}), (\{s_1, \ldots, s_m\}, \{N_1, \ldots, N_l\}))

1. $\Sigma_N \leftarrow \text{distribute}((\{t_1, \ldots, t_n\}, \{N_1, \ldots, N_l\}, k = 0)$ \{List of substitutions for $N_i$\}
2. $\Sigma_M \leftarrow \text{distribute}((\{s_1, \ldots, s_m\}, \{M_1, \ldots, M_k\}, l = 0)$ \{List of substitutions for $M_i$\}
3. $\Sigma \leftarrow \text{map}(\lambda(\sigma_M, \sigma_N).\sigma_M \cup \sigma_N)(\Sigma_M \times \Sigma_N)$
4. return $\Sigma$

Algorithm 5  
\textit{distribute}((\{t_1, \ldots, t_n\}, \{M_1, \ldots, M_k\}), \text{exact})

1. if $n = 0 \land k = 0$ then return $[\{\}]$
3. end if
4. $\Sigma \leftarrow \emptyset$
5. $P_t \leftarrow \text{ordered\_partitions}((\{t_1, \ldots, t_n\}, k)$
6. for $p \in P_t$ do
7. \hspace{1em} $\sigma \leftarrow \{\}$
8. \hspace{1em} for $i \in 1 \to k$ do
9. \hspace{2em} $ts \leftarrow p[i]$
10. \hspace{2em} if $\text{exact}$ then $\sigma \leftarrow \{M_i \mapsto \{ts\|\} \}$
11. \hspace{2em} if $\neg \text{exact} \land ts = \emptyset$ then $\sigma \leftarrow \{M_i \mapsto \{\cdot|M_i\} \}$
12. \hspace{2em} if $\neg \text{exact} \land ts \neq \emptyset$ then $\sigma \leftarrow \{M_i \mapsto \{ts|M_i\} \}$
13. end for
14. $\Sigma \leftarrow \{\sigma\} \cup \Sigma$
15. end for
16. return $\Sigma$

### B Proofs

\textit{Proof for Theorem 3.} We know that $\Sigma_{\text{all}}$ contains at most $n!$ unifiers, one for each way of pairing elements of $\mathcal{I}$ with elements of $\mathcal{P}$. Analogously, $\Sigma$ contains at most $(n - 1)!$ unifiers. Let: $\Sigma_{\text{all}} = \{\sigma_1, \ldots, \sigma_n\}$ and $\Sigma = \{\mu_1, \ldots, \mu_{(n - 1)!}\}$. Then each $\sigma_i$ can be obtained from some $\mu_j$.

If $\sigma_i$ is a unifier resulting from pairing $t_a$ with $s_b$, then there exists $\mu_j = \sigma_i$ and we are done.

Let $\sigma_i$ be a unifier resulting from pairing $t_a$ with some $s_x$, $s_y$ with some $t_y$, and some permutation $P_t$ of $\mathcal{T}\setminus\{t_a, t_y\}$ with some permutation $P_s$ of $\mathcal{T}\setminus\{s_b, s_y\}$. There exists a unifier $\mu_1 \in \Sigma$ that is the result of unifying the same permutations $P_t$ and $P_s$, and $t_y$ with $s_x$. We show how $\sigma_i$ can be reconstructed from $\mu_1$. Since the order in which terms are unified does not matter, we assume that $\sigma_i$ and $\mu_j$ are obtained as follows:
1. Computation of $\sigma_i$:

(a) mgu $\sigma_{ax}$ of $t_a$ and $s_x$  
(b) mgu $\sigma_{by}$ of $t_y\sigma_{ax}$ and $s_t\sigma_{ax}$  
(c) mgu $\sigma_P$ of $P_i\sigma_{ax}\sigma_{by}$ and $P_i\sigma_{ax}\sigma_{by}$

Therefore $\sigma_i = \sigma_{ax}\sigma_{by}\sigma_P$, and $\mu_j = \sigma_{xy}\sigma_P$. We know that:

\[
\begin{align*}
  t_a\sigma_{ax} &= s_x\sigma_{ax} & \text{from 1a} \\
  t_y\sigma_{ax}\sigma_{by} &= s_y\sigma_{ax}\sigma_{by} & \text{from 1b} \\
  t_y\sigma_{ax}\sigma_{by} &= t_a\sigma_{ax}\sigma_{by} & \text{because } t_a = s_b \\
  t_y\sigma_{ax}\sigma_{by} &= s_x\sigma_{ax}\sigma_{by} & \text{from 3 and 1}
\end{align*}
\]

Thus, $\sigma_{ax}\sigma_{by}$ is a unifier of $t_y$ and $s_x$. But from 2a we have that $\sigma_{xy}$ is the most general unifiers of these terms, which means that there exists a $\sigma'$ such that

\[
\sigma_{ax}\sigma_{by} = \sigma_{xy}\sigma'
\]

From 5 and 1c, we know that $P_i\sigma_{xy}\sigma'\sigma_P = P_j\sigma_{xy}\sigma'\sigma_P$, meaning that $\sigma'\sigma_P$ is a unifier of $P_i\sigma_{xy}$ and $P_j\sigma_{xy}$. But from 2b, $\sigma_P$ is the most general unifier of these two lists, therefore, there exists $\sigma''$ such that

\[
\sigma'\sigma_P = \sigma_P\sigma''
\]

Using 5, 6, and associativity of substitution composition:

\[
\sigma_i = \sigma_{ax}\sigma_{by}\sigma_P = \sigma_{xy}\sigma_P\sigma'' = \mu_j\sigma''
\]

\[
\square
\]

**Proof for Theorem 4.** We case on how $t_a$ and $s_b$ were used to compute $\sigma$. Let $t_\rightarrow$ and $s_\rightarrow$ denote the terms and order chosen from $t$ and $s$, respectively, to be unified. Let $t_\rightarrow$ and $s_\rightarrow$ denote the rest of the terms in $t$ and $s$ that will be distributed to $N$ and $M$, respectively.

We know that $\sigma = \sigma_M\sigma_t$ (Alg. 2, line 9), where $\sigma_t$ is the unifier of $t_\rightarrow$ and $s_\rightarrow$, and $\sigma_M$ is obtained from partitions denoted as $\pi_t(t_\rightarrow)$ and $\pi_s(s_\rightarrow)$.

1. $t_a \in t_\rightarrow$ and $s_b \in s_\rightarrow$

   Take $\mu = \mu_M\mu_t$ such that $\mu_M = \sigma_M$ is obtained from the same partitions $\pi_t(t_\rightarrow)$ and $\pi_s(s_\rightarrow)$, and $\mu_t$ is such that $\sigma_i = \mu_t\sigma'$ for some $\sigma'$. The existence of such $\mu_t$ is guaranteed by Theorem 3. Therefore, $\sigma = \sigma_M\sigma_t = \mu_M\mu_t\sigma' = \mu\sigma'$.

2. $t_a \in t_\rightarrow$ and $s_b \in s_\rightarrow$

   Let $t_{p_a}, t_a$ be the part from $\pi_t(t_\rightarrow)$ with $t_a$, and $s_{p_b}, s_b$ the part from $\pi_s(s_\rightarrow)$ with $s_b$.

   Take $\mu = \mu_M\mu_t$ such that $\mu_t = \sigma_1$ is the unifier of $t_{p_a}$ and $s_{p_b}$, and $\mu_M$ is obtained from partitions $\pi_t(t_\rightarrow)$ where $t_{p_a}, t_a$ is replaced by $t_{p_a}$ and analogously for $\pi_s(s_\rightarrow)$. Thus the mappings in $\sigma_M$ and $\mu_M$ are the same, except for two multiset variables $N_a$ and $M_b$:

\[
\begin{align*}
  \{M_b \mapsto \overrightarrow{s_{p_b}}, M_b'\} & : N_a \mapsto t_{p_a}, N_a' \subset \sigma_M \\
  \{M_b \mapsto \overrightarrow{s_{p_b}}, M_b'\} & : N_a \mapsto t_{p_a}, N_a' \subset \mu_M
\end{align*}
\]
Analogous to the previous case.

Since $M'_b$ and $N'_a$ are fresh multiset variable names:

$$\sigma_M = \mu_M\{M'_b \mapsto s_b, M'_b \mapsto t_a, N'_a\}$$

And since $\sigma_t = \mu_t$:

$$\sigma_M\sigma_t = \mu_M\{M'_b \mapsto s_b, M'_b \mapsto t_a, N'_a\}\mu_t$$

The image of $\mu_t$ does not contain $M'_b$ nor $N'_a$, therefore:

$$\sigma_M\sigma_t = \mu_M\mu_t\{M'_b \mapsto s_b, M'_b \mapsto t_a, N'_a\}\mu_t$$

Thus:

$$\sigma = \mu\{M'_b \mapsto s_b, M'_b \mapsto t_a, N'_a\}\mu_t$$

Let $s_k$ be the term from $\overrightarrow{s'_c}$ that is paired with $t_a$ and $\overrightarrow{p_k}$, $s_b$ be the part from $\pi_s(\overrightarrow{r})$ containing $s_b$. Assume that $t_a$ is unified with $s_k$ with mgu $\sigma_{ak}$, and that $(\overrightarrow{t'_c}\langle t_a \rangle)\sigma_{ak}$ unifies with $(\overrightarrow{s'_c}\langle s_k \rangle)\sigma_{ak}$ with mgu $\sigma_c$. Thus $\sigma_t = \sigma_{ak}\sigma_c$. Let $M_b$ be the variable to which partition $\overrightarrow{p_k}, s_b$ is assigned. Thus:

$$\{M_b \mapsto \overrightarrow{p_k}, s_b, M'_b\} \subset \sigma_M$$

By definition

$$\{M_b \mapsto \overrightarrow{p_k}\sigma_{ak}, s_b\sigma_{ak}, M'_b\} \subset \sigma_M\sigma_{ak}$$

Composition with $\sigma_{ak}$

$$\{M_b \mapsto \overrightarrow{p_k}\sigma_{ak}, t_a\sigma_{ak}, M'_b\} \subset \sigma_M\sigma_{ak}$$

$$t_a = s_b$$

$$\{M_b \mapsto \overrightarrow{p_k}\sigma_{ak}, s_k\sigma_{ak}, M'_b\} \subset \sigma_M\sigma_{ak}$$

$$t_a\sigma_{ak} = s_k\sigma_{ak}$$

Take $\mu = \mu_M\mu_t$ such that the terms $\overrightarrow{t'_c}\langle t_a \rangle$ and $\overrightarrow{s'_c}\langle s_k \rangle$ are unified with mgu $\mu_t$, and $\mu_M$ is computed using partition $\pi_t(\overrightarrow{r})$ and $\pi_s(\overrightarrow{r})$ where part $\overrightarrow{p_k}, s_b$ is replaced by $\overrightarrow{p_k}, s_k$. Therefore,

$$\{M_b \mapsto \overrightarrow{p_k}, s_k, M'_b\} \subset \mu_M$$

(5)

Because $\mu_t$ is the mgu of the two lists, we have that: $\sigma_t = \sigma_{ak}\sigma_c = \mu_t\sigma'_t$ for some $\sigma'_t$. And from 4 and 5 we can also conclude: $\sigma_M\sigma_{ak} = \mu_M\sigma_{ak}$. Using these equalities: $\sigma = \sigma_M\sigma_t = \sigma_M\sigma_{ak}\sigma_c = \mu_M\sigma_{ak}\sigma_c = \mu_M\mu_t\sigma'_t = \mu\sigma'_t$.

4. $t_a \in \overrightarrow{r}$ and $s_b \in \overrightarrow{s}$ Analogous to the previous case.