A New Algorithm for Computing Least Generalization of a Set of Atoms

Hien D. Nguyen[1] and Chiaki Sakama[2]

[1] University of Information Technology, VNU-HCM, Vietnam
hiennd@uit.edu.vn
[2] Wakayama University, Japan
sakama@wakayama-u.ac.jp

Abstract. This paper provides a new algorithm of computing a least generalization of a set of atoms. Our algorithm is based on the notion of *anti-combination* that is the inverse substitution of a combined substitution. In contrast to an anti-unification algorithm that computes a least generalization of two atoms, anti-combination can compute a least generalization of (more than two) atoms in parallel. We evaluate the proposed algorithm using randomly generated data and show that anti-combination outperforms the iterative application of an anti-unification algorithm in general.

Keywords: anti-unification · anti-combination · least generalization

1 Introduction

For a definite program P and a goal G, a computed answer θ for $P \cup \{G\}$ is the substitution obtained by restricting the *composition* $\theta_1 \cdots \theta_n$ to the variables of G, where $\theta_1, \ldots, \theta_n$ is the sequence of mgu's used in an SLD-refutation of $P \cup \{G\}$ [10]. In an SLD-derivation, each θ_i is an mgu used for deriving a new goal G_i from its preceding goal G_{i-1} and a parent clause in P. Thus, $\theta_1, \ldots, \theta_n$ are computed sequentially and an answer substitution is computed by composing mgus one by one. As such, the composition operation is not very efficient, and furthermore, it is often unintuitive or inadequate [13]. Yamasaki *et al.* [17] introduce a new resolution method based on *combination* of mgus. The method has the unique feature that resolution is performed by manipulation of substitutions, and mgus used in combination are computed independently of one another. Palamidessi [13] introduces a compositional operational semantics of definite logic programs based on combination of mgus and addresses its application to concurrent logic programming. Compared with the composition operation, however, the combination operation is not well-known and is of relatively little use in automated reasoning and logic programming, although the original idea is date back to [1, 15]. Eder [4] formulates algebraic properties of substitutions and shows that combination is obtained as the greatest lower bound of mgus. In contrast to composition that is only associative, combination is commutative, associative and idempotent, so that it has potential for parallel computation of symbolic reasoning.

In this paper we use the combination operation for computing a *least generalization* (lg) of a set of atoms. Plotkin [14] and Reynolds [16] introduce algorithms for

anti-unification of two atoms. Plotkin argues that the algorithm is iteratively applied to computing a least generalization of a set of atoms: for a set of atoms $\{A_1, \ldots, A_n\}$, its least generalization is computed as $lg(A_1, lg(A_2, \ldots, lg(A_{n-1}, A_n) \cdots))$ where $lg(A_i, A_j)$ computes a least generalization of A_i and A_j. Such serial computation is inefficient when the number of atoms increases. We show that it is computed in parallel using combination of substitutions.

This paper is an extension of the preliminary report [18], which sketches the idea while implementation and evaluation are left. The current paper develops an algorithm and provides experimental evaluation. The rest of this paper is organized as follows. Section 2 reviews basic notions and formal properties of substitutions. Section 3 introduces a method of computing a least generalization by combination of substitutions, and presents an algorithm based on it. Section 4 provides experimental evaluation. Section 5 discusses related issues and Section 6 summarizes the paper.

2 Preliminaries

A *first-order language* consists of an alphabet and all formulas defined over it. The definition is the standard one in the literature [1, 10]. Variables are represented by letters $x, y, z, \ldots$; constants are represented by letters $a, b, c, \ldots$; function symbols (of arities > 0) are represented by letters $f, g, h, \ldots$; and predicate symbols are represented by letters $P, Q, R, \ldots$. A *term* is either (i) a constant, (ii) a variable, or (iii) $f(t_1, \ldots, t_m)$ where f is an m-ary $(m \geq 1)$ function symbol and $t_1, \ldots, t_m$ are terms. An *atom* is a formula $P(t_1, \ldots, t_n)$ $(n \geq 1)$ where P is an n-ary predicate and t_i's are terms. An *expression* is either a term or an atom. Two atoms are *compatible* if they have the same n-ary predicate. The set of all variables (resp. terms, atoms) in the language is denoted by Var (resp. $Term$, $Atom$). The set *Atom* also contains the special elements $\top$ and $\bot$. The set of all expressions is defined as $Exp = Term \cup Atom$. The set of variables occurring in an expression e (resp. a set E of expressions) is denoted by $\mathcal{V}(e)$ (resp. $\mathcal{V}(E)$). The following definitions and results are due to [1, 4, 9, 13, 16].

Definition 1 (substitution). A *substitution* is a mapping σ from Var into $Term$ such that the set $\Gamma = \{ \langle x, \sigma(x) \rangle \mid x \neq \sigma(x) \text{ and } x \in Var \}$ is finite. When $\sigma(x_i) = t_i$ for $i = 1, \ldots, n$, it is also written as $\sigma = \{ t_1/x_1, ..., t_n/x_n \}$.[3] The set of all substitutions in the language is denoted by Sub. The set $\mathcal{D}(\sigma) = \{ x \mid \langle x, t \rangle \in \Gamma \}$ is the *domain* of σ and the set $\mathcal{R}(\sigma) = \{ t \mid \langle x, t \rangle \in \Gamma \}$ is the *range* of σ. The set $\mathcal{V}(\mathcal{R}(\sigma))$ represents the set of all variables occurring in $\mathcal{R}(\sigma)$. The identity mapping ε over Var is the *empty substitution*. A bijection ρ from Var to Var is a *renaming* of variables. The set of all renamings is denoted by Ren (where $Ren \subset Sub$).

Definition 2 ($E\sigma$). Let $\sigma \in Sub$ and $E \in Exp$. Then $E\sigma$ is defined as follows:

$$E\sigma = \begin{cases} \sigma(x) & \text{if } E = x \text{ for } x \in Var, \\ a & \text{if } E = a \text{ for a constant } a, \\ f(t_1\sigma, ..., t_m\sigma) & \text{if } E = f(t_1, ..., t_m) \in Term, \\ P(t_1\sigma, ..., t_n\sigma) & \text{if } E = P(t_1, ..., t_n) \in Atom. \end{cases}$$

[3] It is often written as $\{ x_1/t_1, ..., x_n/t_n \}$ [3, 10, 13].

Definition 3 (composition). For $\sigma, \lambda \in Sub$, the *composition* of σ and λ (denoted by $\sigma\lambda$) is a function from Var to $Term$ such that

$$\sigma\lambda(x) = (x\sigma)\lambda \quad \text{for any } x \in Var.$$

For any $e \in Exp$, it holds that $e(\sigma\lambda) = (e\sigma)\lambda$. The composition operation has the properties: $(\sigma\lambda)\mu = \sigma(\lambda\mu)$ and $\sigma\varepsilon = \varepsilon\sigma = \sigma$ for any $\sigma, \lambda, \mu \in Sub$. Note that $\sigma\lambda \neq \lambda\sigma$ in general.

Definition 4 (idempotent). A substitution σ is *idempotent* if $\sigma\sigma = \sigma$. The set of all idempotent substitutions is denoted by $ISub$.

Proposition 1. ([4, 9]) *A substitution σ is idempotent iff $D(\sigma) \cap \mathcal{V}(\mathcal{R}(\sigma)) = \emptyset$.*

Definition 5 (order on $Atom$). Let $A, B \in Atom$. A preorder relation $\leq$ over $Atom$ is defined as follows:

- $A \leq \top$,
- $\bot \leq A$,
- $A \leq B$ if $A = B\theta$ for some $\theta \in Sub$.

We write $A \sim B$ if $A \leq B$ and $B \leq A$.

When $A \leq B$, we say that A is an *instance* of B (or B is a *generalization* of A). It holds that $A \sim B$ iff $A = B\rho$ for some $\rho \in Ren$. Let $\mathcal{Q}$ be the quotient set $Atom/\sim$. Then the ordered set $(\mathcal{Q}, \leq)$ constitutes a complete lattice [16].

Definition 6 (gci, lcg). Let $\Sigma \subseteq Atom$. An atom $A \in Atom$ is a *common instance* of Σ if $A \leq B$ for any $B \in \Sigma$. In particular, A is a *greatest common instance* (gci) of Σ if A is a common instance of Σ and $A' \leq A$ for any common instance A' of Σ.

An atom $A \in Atom$ is a *common generalization* of Σ if $B \leq A$ for any $B \in \Sigma$. In particular, A is a *least common generalization* (lcg) of Σ if A is a common generalization of Σ and $A \leq A'$ for any common generalization A' of Σ.

Least common generalization is simply called *least generalization* hereafter.

Definition 7 (order on Sub). Let $\sigma, \theta \in Sub$. A preorder relation $\leq$ over Sub is defined as:[4]

$$\sigma \leq \theta \text{ if } \sigma = \theta\lambda \text{ for some } \lambda \in Sub.$$

We write $\sigma \sim \theta$ if $\sigma \leq \theta$ and $\theta \leq \sigma$.

By definition, $\sigma \leq \rho$ for any $\sigma \in Sub$ and $\rho \in Ren$. It holds that $\sigma \sim \theta$ iff $\sigma = \theta\rho$ for some $\rho \in Ren$.

Definition 8 (unifier, mgu, mgsu). Let $\Sigma = \{A_1, \ldots, A_n\}$ be a set of atoms. A substitution $\sigma \in Sub$ is a *unifier* for Σ if $A_1\sigma = \cdots = A_n\sigma$ holds. A unifier σ for a set Σ is a *most general unifier* (mgu) (written $\sigma = mgu(\Sigma)$) if $\theta \leq \sigma$ for any unifier θ for the set Σ. For a finite set $\mathcal{S}$ of finite sets of atoms, $\sigma \in Sub$ is a *most general simultaneous unifier* (mgsu) of $\mathcal{S}$ (written $mgsu(\mathcal{S})$) if $\sigma = mgu(\Sigma)$ for any $\Sigma \in \mathcal{S}$.

[4] We use the same symbol $\leq$ over $Atom$, but the meaning is clear from the context. Note that the relation is often used reversely in the literature, e.g. $\sigma \geq \theta$ if $\sigma = \theta\lambda$ [4].

Proposition 2. ([4, Prop.4.5]) *For any finite set $\Sigma \subseteq Atom$, $\sigma = mgu(\Sigma)$ for some $\sigma \in Sub$ iff there is $\lambda \in ISub$ such that $\lambda = mgu(\Sigma)$ and $\lambda \sim \sigma$.*

By Proposition 2 mgus are assumed to be idempotent in this paper without loss of generality. Let $\mathcal{IS}$ be the quotient set $ISub/\sim$, completed with the bottom element $\bot$. Denote the relation $\leq/\sim$ simply by $\leq$. Then the ordered set $(\mathcal{IS}, \leq)$ constitutes a complete lattice [4].

Definition 9 (combination). For $\Theta \subseteq \mathcal{IS}$, the glb of $(\Theta, \leq)$ is called a *combination*. When $\Theta = \{\theta_1, \ldots, \theta_n\}$, it is written as $\theta_1 + \cdots + \theta_n$.

For any $\sigma,\ \lambda,\ \mu \in ISub$, it holds that (i) $(\sigma + \sigma) \sim \sigma$, (ii) $(\sigma + \lambda) \sim (\lambda + \sigma)$, (iii) $((\sigma + \lambda) + \mu) \sim (\sigma + (\lambda + \mu))$, and (iv) $(\sigma + \varepsilon) \sim \sigma$.

Chang and Lee [1] provide another definition of combination. Given $\theta_1, ..., \theta_n \in ISub$ where $\theta_i = \{t_1^i/x_1^i, ..., t_{k_i}^i/x_{k_i}^i\}$ $(1 \leq i \leq n)$, the combination $\theta_1 + \cdots + \theta_n$ is defined as the mgu of two atoms: $A_1 = P(x_1^1, \ldots, x_{k_1}^1, \ldots, x_1^n, \ldots, x_{k_n}^n)$ and $A_2 = P(t_1^1, \ldots, t_{k_1}^1, \ldots, t_1^n, \ldots, t_{k_n}^n)$. These two definitions are proved to be equivalent [17].[5]

Example 1. ([1, p. 188]) Given $\theta_1 = \{\, f(g(x_1))/x_3, f(x_2)/x_4 \,\}$ and $\theta_2 = \{\, x_4/x_3, g(x_1)/x_2 \,\}$, define $A_1 = P(x_3, x_4, x_3, x_2)$ and $A_2 = P(f(g(x_1)), f(x_2), x_4, g(x_1))$. The mgu of A_1 and A_2 is $\{f(g(x_1))/x_3,\ f(g(x_1))/x_4,\ g(x_1)/x_2\,\}$, which is the combination of θ_1 and θ_2. Note that $\theta_1 + \theta_2 = \theta_1\theta_2$ but $\theta_1 + \theta_2 \neq \theta_2\theta_1 = \{\, f(x_2)/x_3,\ g(x_1)/x_2, f(x_2)/x_4 \,\}$.

The next proposition immediately holds by definition.

Proposition 3. *For $\sigma, \lambda \in ISub$, $\sigma + \lambda = \sigma \cup \lambda$ if $\mathcal{D}(\sigma) \cap \mathcal{D}(\lambda) = \emptyset$.*

Proposition 4. ([4, 17]) *Let $\mathcal{E} = \{\Sigma_1, \ldots, \Sigma_n\}$ be a set of finite sets of atoms.*

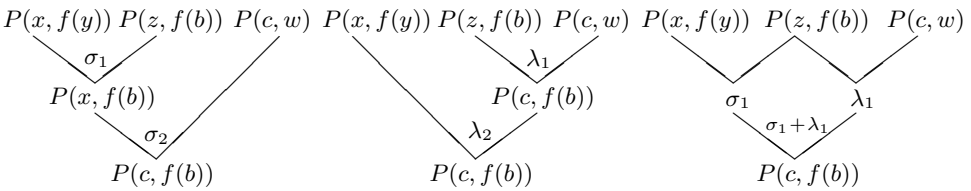
(i) *$mgsu(\mathcal{E}) \sim \sigma_1 \cdots \sigma_n$ where $\sigma_1 = mgu(\Sigma_1)$ and $\sigma_i = mgu(\Sigma_i\sigma_1 \cdots \sigma_{i-1})$ $(2 \leq i \leq n)$.*

(ii) *$mgsu(\mathcal{E}) \sim mgu(\Sigma_1) + \cdots + mgu(\Sigma_n)$.*

Proposition 4 presents two different ways of computing $mgsu(\mathcal{E})$. The one (i) presents that computing $\sigma_1, \ldots, \sigma_n$ in a sequential manner and composing them to get $mgsu(\mathcal{E})$. This method is usually employed in binary resolution. The other one (ii) presents that computing $mgu(\Sigma_i)$ for each Σ_i and combining them to get $mgsu(\mathcal{E})$. Comparing two methods, computation of σ_i uses the results of $\sigma_1, \ldots, \sigma_{i-1}$ in (i). By contrast, in (ii) each $mgu(\Sigma_i)$ is computed independently, so that combination has potential for computing gci in parallel.

Example 2. Consider the set of atoms $\Sigma = \{\, P(x, f(y)),\ P(z, f(b)),\ P(c, w)\,\}$. Let $\Sigma_1 = \{\, P(x, f(y)),\ P(z, f(b))\,\}$ and $\Sigma_2 = \{\, P(z, f(b)),\ f(c, w)\,\}$. Then $\sigma_1 = mgu(\Sigma_1) = \{b/y, x/z\}$ and $\sigma_2 = mgu(\Sigma_2\sigma_1) = \{c/x, f(b)/w\}$. The mgsu of $\{\Sigma_1, \Sigma_2\}$ is then obtained by the composition $\sigma_1\sigma_2 = \{c/x, b/y, c/z, f(b)/w\}$, and the gci of $\Sigma_1 \cup \Sigma_2$ is $P(c, f(b))$ (Fig. 1(a)). Similar computation is done by first computing $\lambda_1 = mgu(\Sigma_2) = \{c/z, f(b)/w\}$ and then computing $\lambda_2 = mgu(\Sigma_1\lambda_1) = \{c/x, b/y\}$, which produces the same mgsu and the gci (Fig. 1(b)). On the other hand, the mgsu is computed by the combination $\sigma_1 + \lambda_1 = \{c/x, b/y, c/z, f(b)/w\}$ which produces the gci (Fig. 1(c)).

[5] Combination is called *parallel composition* in [13].



(a) composition (b) composition (c) combination

Fig. 1: composition and combination

3 Computing Least Generalization by Anti-combination

3.1 Anti-unification algorithm

For a set $\Sigma \subseteq Atom$, its *least (common) generalization* (written $lg(\Sigma)$) is defined as the least upper bound of the set $(\Sigma, \leq)$ (Def. 6). $lg(\Sigma)$ is obtained from Σ by anti-unification, that is a dual of unification.

Definition 10 (anti-unifier, msau). ([11]) Let $\Sigma = \{A_1, \ldots, A_k\}$ be a set of atoms. Then, a tuple of substitutions $\tau = (\sigma_1, \ldots, \sigma_k)$ where $\sigma_i \in Sub$ $(1 \leq i \leq k)$ is an *anti-unifier* of Σ if $A_i = lg(\Sigma)\sigma_i$ for $i = 1, \ldots, k$. An anti-unifier τ of Σ is a *most specific anti-unifier* (msau) if for each anti-unifier $(\theta_1, \ldots, \theta_k)$ there is a substitution $\lambda_i \in Sub$ such that $\sigma_i = \lambda_i\theta_i$ $(1 \leq i \leq k)$. We define $\mathcal{D}(\tau) = \mathcal{D}(\sigma_1) \cup \cdots \cup \mathcal{D}(\sigma_k)$.

An anti-unifier always exists, but is not necessarily unique. There is a unique most specific anti-unifier that produces the least generalization, which is unique up to renaming of variables [11]. Like an mgu, an msau is also assumed to be idempotent.

Proposition 5. *Let Σ be a set of atoms. Then $(\sigma_1, \ldots, \sigma_k)$ such that $\sigma_i \in Sub$ $(1 \leq i \leq k)$ is an msau of Σ iff there is an msau $(\lambda_1, \ldots, \lambda_k)$ such that $\lambda_i \in ISub$ and $\lambda_i \sim \sigma_i$ for $i = 1, \ldots, k$.*

Proof. When $\mathcal{D}(\sigma_i) \cap \mathcal{V}(\mathcal{R}(\sigma_i)) \neq \emptyset$, let $\lambda_i = \sigma_i\rho_i$ where $\rho_i \in Ren$ and $\mathcal{D}(\lambda_i) \cap \mathcal{V}(\mathcal{R}(\lambda_i)) = \emptyset$. In this case, $A_i = lg(\Sigma)\sigma_i$ implies $A_i \sim lg(\Sigma)\lambda_i$, and vice-versa. □

Now we recall the *anti-unification algorithm* [3, Algorithm 13.1] for computing a least generalization of two atoms which is originally introduced in [14, 16]. Given an atom $A = P(t_1, \ldots, t_n)$, a term t_i $(1 \leq i \leq n)$ has *position* $\langle i \rangle$ in A. If a term $f(s_1, \ldots, s_m)$ has position $\langle p_1, \ldots, p_k \rangle$ in A, then s_j within this term has position $\langle p_1, \ldots, p_k, j \rangle$ in A. The algorithm is described in Figure 2.

Since the lub of $(\Sigma, \leq)$ is associative, the anti-unification algorithm is iteratively applied for computing a least generalization of a set Σ of atoms. In this case, an anti-unifier is computed by a composition of substitutions.

Example 3. Let $\Sigma = \{A_1, A_2, A_3\}$, $G_1 = lg(\{A_1, A_2\})$ and $G_2 = lg(\{A_1, A_2, A_3\}) = lg(\{G_1, A_3\})$. Then $A_1 = G_1\theta_1$, $A_2 = G_1\theta_2$, $G_1 = G_2\sigma_1$, and $A_3 = G_2\sigma_2$ for some θ_1, θ_2, σ_1, $\sigma_2 \in Sub$. Then $A_1 = G_2\sigma_1\theta_1$, $A_2 = G_2\sigma_1\theta_2$, and $A_3 = G_2\sigma_2$. So $(\sigma_1\theta_1, \sigma_1\theta_2, \sigma_2)$ is an anti-unifier of (A_1, A_2, A_3).

Input : Two compatible atoms A_1 and A_2
Output : $G = lg(\{A_1, A_2\})$ and an msau $\tau = (\theta_1, \theta_2)$

1. Set $A'_1 = A_1$ and $A'_2 = A_2$, $\theta_1 = \theta_2 = \varepsilon$, and $i = 0$.
   Let $z_1, z_2, \ldots$ be a sequence of variables not appearing in A_1 or A_2.
2. If $A'_1 = A'_2$, then output $G := A'_1$, $\tau := (\theta_1, \theta_2)$ and stop.
3. Let p be the leftmost symbol position where A'_1 and A'_2 differ. Let s and t be the terms occurring at this position in A'_1 and A'_2, respectively.
4. If, for some j with $1 \leq j \leq i$, $z_j\theta_1 = s$ and $z_j\theta_2 = t$, then replace s at the position p in A'_1 by z_j, replace t at the position p in A'_2 by z_j, and go to 2.
5. Otherwise set i to $i + 1$, replace s at the position p in A'_1 by z_i, and replace t at the position p in A'_2 by z_i. Set θ_1 to $\theta_1 \cup \{s/z_i\}$, θ_2 to $\theta_2 \cup \{t/z_i\}$, and go to 2.

Fig. 2: **Anti-unification Algorithm [3]**

The above algorithm computes a substitution θ_i such that $A_i = G\theta_i$ $(i = 1, 2)$ for $G = lg(\{A_1, A_2\})$. Then an lg G is also computed by $G = A_i\theta_i^{-1}$ where θ_i^{-1} is an *inverse substitution* of θ_i. An inverse substitution θ^{-1} is well-defined if θ is injective.

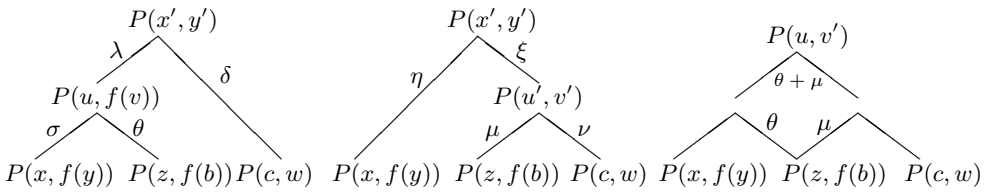
Definition 11 (inverse substitution). ([12]) Let $\theta \in Sub$ be injective and $t \in Term$. If $\mathcal{D}(\theta) \cap \mathcal{V}(t) = \emptyset$, then an *inverse substitution* $\theta^{-1} : Term \to Var$ is defined as follows.

$$
\begin{aligned}
t\theta^{-1} &= x \qquad \text{if } (t/x) \in \theta, \\
f(t_1, \ldots, t_n)\theta^{-1} &= f(t_1\theta^{-1}, \ldots, t_n\theta^{-1}) \text{ if } (f(t_1, \ldots, t_n)/x) \notin \theta \text{ for any } x \in Var, \\
y\theta^{-1} &= y \qquad \text{if } (y/x) \notin \theta \text{ for any } x \in Var.
\end{aligned}
$$

If $\mathcal{D}(\theta) \cap \mathcal{V}(t) \neq \emptyset$, a renaming substitution $\rho \in Ren$ is applied to t in such a way that $\mathcal{D}(\theta) \cap \mathcal{V}(t\rho) = \emptyset$. Then we can apply θ^{-1} to $t\rho$ if θ is injective. If a substitution θ is not injective, we use the technique of [3] to constitute θ^{-1}. For instance, when $t = f(x, y)$ and $\theta = \{a/x, a/y\}$, it becomes $t\theta = f(a, a)$. The inverse substitution $\theta^{-1} = \{x/a, y/a\}$ is ill-defined, then it is modified as $\theta^{-1} = \{(x/a, \langle\, 1 \,\rangle), (y/a, \langle\, 2 \,\rangle)\}$ meaning that a at position $\langle\, 1 \,\rangle$ is mapped to x and a at position $\langle\, 2 \,\rangle$ is mapped to y. With this mechanism, $f(a, a)\theta^{-1} = f(x, y)$. For any non-injective $\theta \in Sub$, we constitute θ^{-1} in this way.

Definition 12 (anti-combination). Let $\sigma = \theta_1 + \cdots + \theta_n$ be a combination of $\theta_i \in ISub$ $(1 \leq i \leq n)$. Then the inverse substitution σ^{-1} is called an *anti-combination* of $\theta_1, \ldots, \theta_n$.

Combining injective substitutions may produce a non-injective substitution. For instance, $\theta_1 = \{a/x\}$ and $\theta_2 = \{a/y\}$ produce $\theta_1 + \theta_2 = \{a/x, a/y\}$. To compute its inverse substitution, we incorporate information of substitutions from which each binding comes from: $(\theta_1 + \theta_2)^{-1} = \{(x/a, \langle\, \theta_1 \,\rangle), (y/a, \langle\, \theta_2 \,\rangle)\}$ which means that a from



(a) anti-unification (b) anti-unification (c) anti-combination

Fig. 3: anti-unification and anti-combination

θ_1 is mapped to x and a from θ_2 is mapped to y. With this technique, anti-combination is well-defined for non-injective combination.

Lemma 1. *Let $\Sigma = \{A_1, A_2, A_3\}$ be a set of atoms, $\tau_{12} = (\sigma_{12}, \lambda_{12})$ an msau of $\{A_1, A_2\}$, and $\tau_{13} = (\sigma_{13}, \lambda_{13})$ an msau of $\{A_1, A_3\}$ such that $\mathcal{D}(\tau_{12}) \cap \mathcal{D}(\tau_{13}) = \emptyset$. Then $lg(\Sigma) = A_1\theta^{-1}$ where $\theta \sim (\sigma_{12} + \sigma_{13})$.*

Proof. Let $G_1 = lg(\{A_1, A_2\})$ and $G_2 = lg(\{A_1, A_3\})$. Then $lg(\Sigma) = lg(\{G_1, G_2\})$, and $G_1\sigma_{12} = A_1$ and $G_2\sigma_{13} = A_1$. By $\mathcal{D}(\sigma_{12}) \cap \mathcal{D}(\sigma_{13}) = \emptyset$, $G_1\sigma_{12} = G_1(\sigma_{12} + \sigma_{13}) = A_1$ and $G_2\sigma_{13} = G_2(\sigma_{12} + \sigma_{13}) = A_1$. Then $lg(\Sigma) = lg(\{G_1, G_2\}) = lg(\{A_1(\sigma_{12} + \sigma_{13})^{-1}, A_1(\sigma_{12} + \sigma_{13})^{-1}\}) = A_1(\sigma_{12} + \sigma_{13})^{-1}$. □

Since combination is associative, the result of Lemma 1 is extended to a set containing n atoms ($n \geq 3$).

Theorem 1. *Let $\Sigma = \{A_1, \ldots, A_n\}$ $(n \geq 3)$ be a set of atoms, $\tau_{1k} = (\sigma_{1k}, \lambda_{1k})$ $(2 \leq k \leq n)$ an msau of $\{A_1, A_k\}$ such that $\mathcal{D}(\tau_{1i}) \cap \mathcal{D}(\tau_{1j}) = \emptyset$ $(1 \leq i, j \leq n; i \neq j)$. Then, $lg(\Sigma) = A_1\theta^{-1}$ where $\theta \sim (\sigma_{12} + \cdots + \sigma_{1n})$.*

Theorem 1 shows that a least generalization of atoms is computed by anti-combination of substitutions.

Example 4. Consider the set $\Sigma = \{ P(x, f(y)),\ P(z, f(b)),\ P(c, w) \}$ of atoms. Then $lg(\{ P(x, f(y)),\ P(z, f(b)) \}) = P(u, f(v))$ with the msau (σ, θ) where $\sigma = \{x/u,\ y/v\}$ and $\theta = \{z/u,\ b/v\}$. In this case, $P(u, f(v))\sigma = P(x, f(y))$ and $P(u, f(v))\theta = P(z, f(b))$.

Next, $lg(\{P(u, f(v)), P(c, w)\}) = P(x', y')$ with the msau (λ, δ) where $\lambda = \{u/x', f(v)/y'\}$ and $\delta = \{c/x', w/y'\}$. In this case, $P(x', y')\lambda = P(u, f(v))$ and $P(x', y')\delta = P(c, w)$. Then, $lg(\Sigma) = P(x', y')$ where $P(x', y')\lambda\sigma = P(x, f(y))$ with $\lambda\sigma = \{x/x', f(y)/y'\}$ and $P(x', y')\lambda\theta = P(z, f(b))$ with $\lambda\theta = \{z/x', f(b)/y'\}$ (Fig. 3(a)). Similar computation is done by first computing $lg(\{ P(z, f(b)), P(c, w)\})$ with $(\mu, \nu) = (\{z/u', f(b)/v'\},\ \{c/u', w/v'\})$, and then computing $lg(\{ P(x, f(y)),\ lg(\{ P(z, f(b)), P(c, w)\}) \})$ with $(\eta, \xi) = (\{x/x', f(y)/y'\}, \{u'/x', v'/y'\})$. (Fig. 3(b)).

By contrast, $\theta + \mu = \{z/u, b/v, z/u', f(b)/v'\}$. Then

$$(\theta + \mu)^{-1} = \{\, (u/z, \langle\, \theta \,\rangle),\ (v/b, \langle\, \theta \,\rangle),\ (u'/z, \langle\, \mu \,\rangle),\ (v'/f(b), \langle\, \mu \,\rangle) \,\}.$$

Applying it to $P(z, f(b))$, $lg(\Sigma) = P(u, v')$ $(\sim P(x', y'))$ is obtained (Fig. 3(c)). Note that by the second condition of Def. 11, $(v/b, \langle\, \theta \,\rangle)$ is not applied to b in $P(z, f(b))$.

3.2 Algorithms for computing least generalization of a set of atoms

The algorithm for computing anti-unification (Fig. 2) is extended to computing a least generalization of a set of atoms. Given a set Σ, $\Sigma[i]$ means the i-th element of Σ.

Algorithm 1: AntiUnif

Input : A set $\Sigma = \{A_1, \ldots, A_n\}$ $(n \geq 2)$ of compatible atoms
Output : a least generalization of Σ

1. Put $G := \Sigma[1]$.
2. Put $i := 2$; while $i \leq n$ do:
   Compute $G := lg(\{G, \Sigma[i]\})$ by the anti-unification algorithm (Fig. 2).
   Put $i := i + 1$.
3. Return G.

The algorithm for computing a least generalization of a set of atoms by anti-combination is described as follows.

Algorithm 2: AntiComb

Input : A set $\Sigma = \{A_1, \ldots, A_n\}$ $(n \geq 2)$ of compatible atoms
Output : a least generalization of Σ

1. Put $\theta := \varepsilon$ (empty substitution).
2. Put $i := 2$; while $i \leq n$ do:
   Compute $G_i = lg(\{A_1, A_i\})$ by the anti-unification algorithm.
   Get a substitution θ_i such that $A_1 = G_i\theta_i$, $\mathcal{D}(\theta_i) \cap \mathcal{D}(\theta) = \emptyset$ and
   $\mathcal{D}(\theta_i) \cap \mathcal{V}(\mathcal{R}(\theta_i)) = \emptyset$.
   Put $\theta := \theta + \theta_i$ and $i := i + 1$.
3. Compute the inverse substitution θ^{-1}.
4. Compute $G = A_1\theta^{-1}$ and return G.

In θ_i we store information about the substitution and the position of each element as $[t_i/z_i, \langle\, p, q \,\rangle, \theta_i]$, meaning that t_i/z_i in θ_i happens at the q-th position of the p-th arity.

When $k\,(2 \leq k < n)$ processors are available, the step 2 of Algorthm 2 is split into k procedures. First, Σ is partitioned into k subsets $\Sigma_1 \cup \cdots \cup \Sigma_k$ such that $\Sigma_1 = \{A_1, \ldots, A_{m_1}\}, \Sigma_2 = \{A_{m_1}, A_{m_1+1}, \ldots, A_{m_2}\}, \ldots, \Sigma_k = \{A_{m_{k-1}}, A_{m_{k-1}+1}, \ldots, A_{m_k}\}$ where each Σ_i and Σ_{i+1} share an element A_{m_i} in common. After computing a combination θ^i for each Σ_i $(1 \leq i \leq k)$ in parallel, they are combined into one substitution $\theta = \theta^1 + \cdots + \theta^k$. Then, its inverse substitution is computed at the step 3. Formally, such a spliting is done by

$$\Sigma_j = \{\Sigma[(j-1) \times \lfloor \frac{n}{k} \rfloor + 1], \ldots, \Sigma[j \times \lfloor \frac{n}{k} \rfloor + 1]\} \;\; (1 \leq j \leq k-1),$$
$$\Sigma_k = \{\Sigma[(k-1) \times \lfloor \frac{n}{k} \rfloor + 1], \ldots, \Sigma[n]\}$$

where $\lfloor\ \rfloor$ is the floor function.

According to [7] the complexity of the anti-unification algorithm of Fig. 2 is computed in $O(N \log N)$ where N is the size of the lub of θ_1 and θ_2. Using the result, the complexity of **AntiUnif** is $O(n \times N \log N)$ where n is the number of atoms in Σ. Step 2 of **AntiComb** is also done in $O(n \times N \log N)$, since combination of substitution is computed by merging $\theta \cup \theta_i$ (Proposition 3). If k-processors $(k \geq 2)$ are available for computing Step 2 in parallel, the lower bound of computation is given as $O(\frac{n \times N \log N}{k})$.

Example 5. Suppose the set Σ of atoms such that $|\Sigma| = 5$ and each atom has a ternary predicate P.[6]

$E := \{$
$A_1 = P(\underline{f_9(x_2, x_2, x_1)}, \underline{f_{10}(x_2, f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1))))}, \underline{x_1}),$
$A_2 = P(\underline{f_9(x_2, x_2, x_1)}, \underline{f_{10}(f_3(f_4(f_8(x_2, x_1), x_2, f_7(x_2, x_1))), f_1(x_1, x_1, x_1))}, \underline{f_3(x_2)}),$
$A_3 = P(\underline{f_9(x_2, x_2, x_1)}, \underline{f_{10}(f_5(x_1), f_5(f_5(f_6(x_2, x_1))))}, \underline{f_3(f_{10}(x_2, x_1))}),$
$A_4 = P(\underline{f_9(f_1(x_1, x_2, x_2), x_1, f_2(x_1, x_1, x_1))}, \underline{f_{10}(f_7(f_9(f_6(x_1, x_1), f_6(x_1, x_2), x_1), x_1), x_1)}, \underline{x_1}),$
$A_5 = P(\underline{f_9(f_4(x_2, x_2, x_1), f_4(x_2, x_1, x_2), f_{10}(x_1, x_1))},$
$\underline{f_{10}(x_2, f_4(f_5(f_9(x_1, x_2, x_1)), x_2, f_5(f_5(x_1))))}, \underline{x_2})$
$\}.$

Using **AntiUnif** :

$G_1 := lg(A_1, A_2) = P(f_9(x_2, x_2, x_1), f_{10}(z_1, z_2), z_3)$ % **Compute** $lg(A_1, A_2)$
$G_2 := lg(G_1, A_3) = P(f_9(x_2, x_2, x_1), f_{10}(z_4, z_5), z_6)$ % **Compute** $lg(G_1, A_3)$
$G_3 := lg(G_2, A_4) = P(f_9(z_7, z_8, z_9), f_{10}(z_{10}, z_{11}), z_{12})$ % **Compute** $lg(G_2, A_4)$
$G_4 := lg(G_3, A_5) = P(f_9(z_{13}, z_{14}, z_{15}), f_{10}(z_{16}, z_{17}), z_{18})$ % **Compute** $lg(G_3, A_5)$

Using **AntiComb** :

$G_1 := lg(A_1, A_2) = P(f_9(x_2, x_2, x_1), f_{10}(z_1, z_2), z_3)$ % **Compute** $lg(A_1, A_2)$
$\theta_1 := \{ [x_2/z_1, \langle 2, 1 \rangle, \theta_1], [f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_2, \langle 2, 2 \rangle, \theta_1],$
$[x_1/z_3, \langle 3 \rangle, \theta_1] \};$
$G_2 := lg(A_1, A_3) = P(f_9(x_2, x_2, x_1), f_{10}(z_4, z_5), z_6)$ % **Compute** $lg(A_1, A_3)$
$\theta_2 := \{ [x_2/z_4, \langle 2, 1 \rangle, \theta_2], [f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_5, \langle 2, 2 \rangle, \theta_2],$
$[x_1/z_6, \langle 3 \rangle, \theta_2] \};$
$G_3 := lg(A_1, A_4) = P(f_9(z_7, z_8, z_9), f_{10}(z_{10}, z_{11}), x_1)$ % **Compute** $lg(A_1, A_4)$
$\theta_3 := \{ [x_2/z_7, \langle 1, 1 \rangle, \theta_3], [x_2/z_8, \langle 1, 2 \rangle, \theta_3], [x_1/z_9, \langle 1, 3 \rangle, \theta_3], [x_2/z_{10}, \langle 2, 1 \rangle, \theta_3],$
$[f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_{11}, \langle 2, 2 \rangle, \theta_3] \};$
$G_4 := lg(A_1, A_5) = P(f_9(z_{12}, z_{13}, z_{14}), f_{10}(x_2, z_{15}), z_{16})$ % **Compute** $lg(A_1, A_5)$
$\theta_4 := \{ [x_2/z_{12}, \langle 1, 1 \rangle, \theta_4], [x_2/z_{13}, \langle 1, 2 \rangle, \theta_4], [x_1/z_{14}, \langle 1, 3 \rangle, \theta_4],$
$[f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_{15}, \langle 2, 2 \rangle, \theta_4], [x_1/z_{16}, \langle 3 \rangle, \theta_4] \};$

[6] Here we draw underlines to help distinguishing 3 terms in P.

$\theta := \theta_1 + \theta_2 + \theta_3 + \theta_4$ % **Compute combination**
$= \{\, [x_2/z_1, \langle\, 2, 1 \,\rangle, \theta_1],\ [f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_2, \langle\, 2, 2 \,\rangle, \theta_1],$
$[x_1/z_3, \langle\, 3 \,\rangle, \theta_1],\ [x_2/z_4, \langle\, 2, 1 \,\rangle, \theta_2],\ [f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_5, \langle\, 2, 2 \,\rangle, \theta_2],$
$[x_1/z_6, \langle\, 3 \,\rangle, \theta_2],\ [x_2/z_7, \langle\, 1, 1 \,\rangle, \theta_3], [x_2/z_8, \langle\, 1, 2 \,\rangle, \theta_3],\ [x_1/z_9, \langle\, 1, 3 \,\rangle, \theta_3],$
$[x_2/z_{10}, \langle\, 2, 1 \,\rangle, \theta_3],\ [f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_{11}, \langle\, 2, 2 \,\rangle, \theta_3],$
$[x_2/z_{12}, \langle\, 1, 1 \,\rangle, \theta_4],\ [x_2/z_{13}, \langle\, 1, 2 \,\rangle, \theta_4],\ [x_1/z_{14}, \langle\, 1, 3 \,\rangle, \theta_4],$
$[f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1)))/z_{15}, \langle\, 2, 2 \,\rangle, \theta_4],\ [x_1/z_{16}, \langle\, 3 \,\rangle, \theta_4] \,\};$

$\theta^{-1} :=$ % **Compute anti-combination**
$\{\, [z_1/x_2, \langle\, 2, 1 \,\rangle, \theta_1], [z_2/f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1))), \langle\, 2, 2 \,\rangle, \theta_1],$
$[z_3/x_1, \langle\, 3 \,\rangle, \theta_1],\ [z_4/x_2, \langle\, 2, 1 \,\rangle, \theta_2],\ [z_5/f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1))), \langle\, 2, 2 \,\rangle, \theta_2],$
$[z_6/x_1, \langle\, 3 \,\rangle, \theta_2],\ [z_7/x_2, \langle\, 1, 1 \,\rangle, \theta_3],\ [z_8/x_2, \langle\, 1, 2 \,\rangle, \theta_3],\ [z_9/x_1, \langle\, 1, 3 \,\rangle, \theta_3],$
$[z_{10}/x_2, \langle\, 2, 1 \,\rangle, \theta_3],\ [z_{11}/f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1))), \langle\, 2, 2 \,\rangle, \theta_3],$
$[z_{12}/x_2, \langle\, 1, 1 \,\rangle, \theta_4],\ [z_{13}/x_2, \langle\, 1, 2 \,\rangle, \theta_4],\ [z_{14}/x_1, \langle\, 1, 3 \,\rangle, \theta_4],$
$[z_{15}/f_7(f_5(f_{10}(x_1, x_1)), f_7(f_2(x_2, x_2, x_1), f_3(x_1))), \langle\, 2, 2 \,\rangle, \theta_4],\ [z_{16}/x_1, \langle\, 3 \,\rangle, \theta_4] \,\};$

$A_1\theta^{-1} := P(f_9(z_{12}, z_{13}, z_{14}), f_{10}(z_{10}, z_{15}), z_{16}).$ % **Compute least generalization**

When there are different replacements z_i/t_i from different θ_j's at the same position $\langle\, m\,, n \,\rangle$ in θ^{-1}, for instance, $[z_7/x_2, \langle\, 1, 1 \,\rangle, \theta_3]$ and $[z_{12}/x_2, \langle\, 1, 1 \,\rangle, \theta_4]$, they are equivalent modulo variable renaming and one of them is selected.

4 Experimental Evaluation

In this section, we compare runtime for computing least generalizations by two algorithms **AntiUnif** and **AntiComb**.

4.1 Generating Test Data

We use randomly created data sets $Prog$ satisfying the following conditions.

1. Each element in $Prog$ is an atom of the form: $P(t_1, t_2, t_3)$ where P is a ternary predicate and t_1, t_2, t_3 are terms. Every atom in $Prog$ has the same predicate.
2. $Prog$ has two parameters: n is the number of elements in $Prog$, and m is the number of function symbols appearing in $Prog$. The number of different variables appearing in $Prog$ is set to $\frac{n}{2}$, while there is no constant in $Prog$.
3. For an atom $A = P(t_1, t_2, t_3)$, the *depth* of A is defined as $d(A) = 1 + max\{d(t_1), d(t_2), d(t_3)\}$ where $d(t_i)$ $(1 \leq i \leq 3)$ is the number of function symbols appearing in t_i. For instance, the depth of $P(x, y, z)$ is 1, the depth of $P(f(x), y, g(h(z)))$ is 3, and so on. We set the depth of each atom A in $Prog$ as $d(A) \leq 5$.
4. For any atom $P(t_1, t_2, t_3)$ in $Prog$, if a function f appears in the outermost of the term t_i $(1 \leq i \leq 3)$, then the outermost function appearing in the corresponding term s_i of another atom $P(s_1, s_2, s_3)$ in $Prog$ is set to the same function f.

An explanation is in order for the above 4th condition on $Prog$. For instance, two atoms: $P(f(g(x)), y, g(z))$ and $P(f(h(x)), g(y), g(h(z)))$ have the same outermost function f in terms appearing in the 1st arity of P and the same outermost function g in the 3rd arity. This is because we randomly generate a set $Prog$ then it is very unlikely that the same function appears in the corresponding positions of more than two elements. Without this assumption, computation of anti-unification will be simple and the results are likely to contain no function. For instance, the result of anti-unification of $P(f_1(g_1(x)), y, g_2(z))$ and $P(f_2(h_1(x)), g_3(y), g_4(h_2(z)))$ becomes $P(x, y, z)$. Note that it may happen that no function appears in the i-th arity $(1 \leq i \leq 3)$ of P as in the 2nd element of $P(f(g(x)), y, g(z))$.

4.2 Experimental Results

We compare runtime for computing a least generalization of $Prog$ using **AntiUnif** and **AntiComb**. We implement two algorithms by Maple 2018, 64 bit. The testing is done on a computer with the following configuration: Intel(R) Core^{TM} i7-4750HQ CPU@ 2.0 GHz, RAM 8.00GB, Operating system: Windows 10, 64-bit.

In the experiments, we set the parameters n and m as follows:

- The number of atoms in $Prog$ is set to: $n = 500, 1000, 3000, 5000, 10000$.
- The number of functions appearing in $Prog$ is set to: $m = n/2$; $m = n$; $m = 2n$.

Based on (n, m), generate the set of atoms $Prog$ randomly. In **AntiComb**, the number of processors k is set to $k = 10; 30; 50$. For each (n, m, k) we measure runtime at least four times and pick average values.

In experiments, we do not have many computers for parallel computing. So we compute the time for k-parallel processing by $max\{t_1, \ldots, t_k\}$ where t_i $(1 \leq i \leq k)$ is the time for computing a combination θ^i for Σ_i. After computing each θ^i, they are combined into $\theta = \theta^1 + \cdots + \theta^k$. This process is denoted by Stage 1. After Stage 1, produce the inverse θ^{-1} and compute $A_1\theta^{-1}$ that is the least generalization $lg(\Sigma)$ of the input set Σ. This process is denoted by Stage 2.

Table 1 shows the experimental results that are displayed in Figures 4 and 5.

By the results of testing, it is observed that **AntiComb** is faster than **AntiUnif** in general. This is because **AntiComb** can compute least generalization in parallel. The time of computing a least generalization by **AntiComb** decreases by increasing the number of processors for parallel computing. Note that runtime T_{AC} for **AntiComb** is greater than the value T_{AU}/k where T_{AU} is runtime for **AntiUnif**. This is because the relation $T_{AU}/k < T_{AC} < T_{AU}/k + T_\infty$ holds by *Brent's law* [6], where T_∞ is runtime using an idealized machine with an infinite number of processors. It is known that parallel computing is effective when the number of processors is small, or when the problem is perfectly parallel (*Amdahl's law*) [5]. In **AntiComb**, Stage 1 is (partly) computed in parallel while Stage 2 is serial. Hence, the *speedup* of the algorithm $S = T_{AU}/T_{AC}$ is limited by the time needed for the serial fraction of the problem. The *efficiency* of parallel computing $E = S/k$ also decreases by increasing k. Moreover, we compute runtime for parallel processing by $max\{t_1, \ldots, t_k\}$. These factors make the speedup of **AntiComb** seemingly smaller than the number of processors used.

Table 1: Experimental Results

n	m	AntiUnif	AntiComb (sec)			
		(sec)	k	Stage 1	Stage 2	Total
500	250	0.203	10	0.047	0.031	0.078
			30	0.032	0.031	0.063
			50	0.016	0.031	0.047
500	500	0.218	10	0.032	0.015	0.047
			30	0.016	0.031	0.047
			50	0.016	0.016	0.032
500	1000	0.406	10	0.141	0.031	0.172
			30	0.031	0.047	0.078
			50	0.016	0.047	0.063

n	m	AntiUnif	AntiComb (sec)			
		(sec)	k	Stage 1	Stage 2	Total
1000	500	0.406	10	0.063	0.25	0.313
			30	0.032	0.062	0.094
			50	0.032	0.047	0.079
1000	1000	0.422	10	0.266	0.078	0.344
			30	0.016	0.078	0.094
			50	0.016	0.062	0.078
1000	2000	0.61	10	0.063	0.063	0.126
			30	0.032	0.062	0.094
			50	0.016	0.047	0.063

n	m	AntiUnif	AntiComb (sec)			
		(sec)	k	Stage 1	Stage 2	Total
3000	1500	0.719	10	0.281	0.093	0.374
			30	0.219	0.078	0.297
			50	0.125	0.078	0.203
3000	3000	0.922	10	0.359	0.093	0.452
			30	0.047	0.109	0.156
			50	0.016	0.094	0.11
3000	6000	0.953	10	0.437	0.109	0.546
			30	0.219	0.11	0.329
			50	0.172	0.094	0.266

n	m	AntiUnif	AntiComb (sec)			
		(sec)	k	Stage 1	Stage 2	Total
5000	2500	1.218	10	0.547	0.125	0.672
			30	0.297	0.141	0.438
			50	0.276	0.125	0.401
5000	5000	2.125	10	0.829	0.204	1.033
			30	0.5	0.234	0.734
			50	0.328	0.219	0.547
5000	10000	2.891	10	1.109	0.234	1.343
			30	0.641	0.25	0.891
			50	0.453	0.234	0.687

n	m	AntiUnif	AntiComb (sec)					
		(sec)	k	Stage 1	Stage 2	Total	Speedup †	Efficiency ‡
10000	5000	1.641	10	1.313	0.141	1.454	1.13	0.11
			30	0.515	0.125	0.64	2.56	0.09
			50	0.359	0.125	0.484	3.39	0.07
10000	10000	2.359	10	1.047	0.187	1.234	1.91	0.19
			30	0.547	0.188	0.735	3.21	0.11
			50	0.485	0.203	0.688	3.43	0.07
10000	20000	2.953	10	1.406	0.266	1.672	1.77	0.18
			30	0.579	0.297	0.876	3.37	0.11
			50	0.422	0.282	0.704	4.19	0.08

† Speedup:= AntiUnif / AntiComb(Total)
‡ Efficiency:=Speedup / k

5 Discussion

Palamidessi [13] uses the least upper bound (that is called the glb in the context of [13]) of substitutions for *parallel factorization* that corresponds to least generalization. Given two substitutions θ and σ with different bindings for the same variable, say $t/x \in \theta$ and $u/x \in \sigma$, she eliminates the difference by replacing t and u by x in θ and σ respectively. For instance, given two substitutions $\theta = \{\, a/x, f(a)/y \,\}$ and $\sigma = \{b/x, f(b)/y\}$, $\lambda = \{f(x)/y\}$ is computed as the least upper bound of θ and σ by replacing a and b by x. This corresponds to computing a least generalization of two atoms $A_1 = P(a, f(a))$ and $A_2 = P(b, f(b))$ using the anti-unification algorithm, which outputs $lg(\{A_1, A_2\}) = P(x, f(x))$ and the msau $(\{a/x\}, \{b/x\})$. In the anti-combination algorithm **AntiComb**, on the other hand, given the set of n atoms $\{A_1, \ldots, A_n\}$ such that $A_1 = P(a, f(a))$ and $A_2 = P(b, f(b))$, $lg(\{A_1, A_2\}) = P(x, f(x))$ and the substitution $\theta_2 = \{a/x\}$ is computed. θ_2 is then combined with other substitution θ_i such that $A_1 = G_i\theta_i$ and $G_i = lg(\{A_1, A_i\})$ for $3 \leq i \leq n$. As such, anti-combination is different from parallel factorization. In fact, the parallel factorization algorithm introduced in [13] outputs the lub of two substitutions, in other words, it computes anti-unification of two terms by manipulating substitutions.

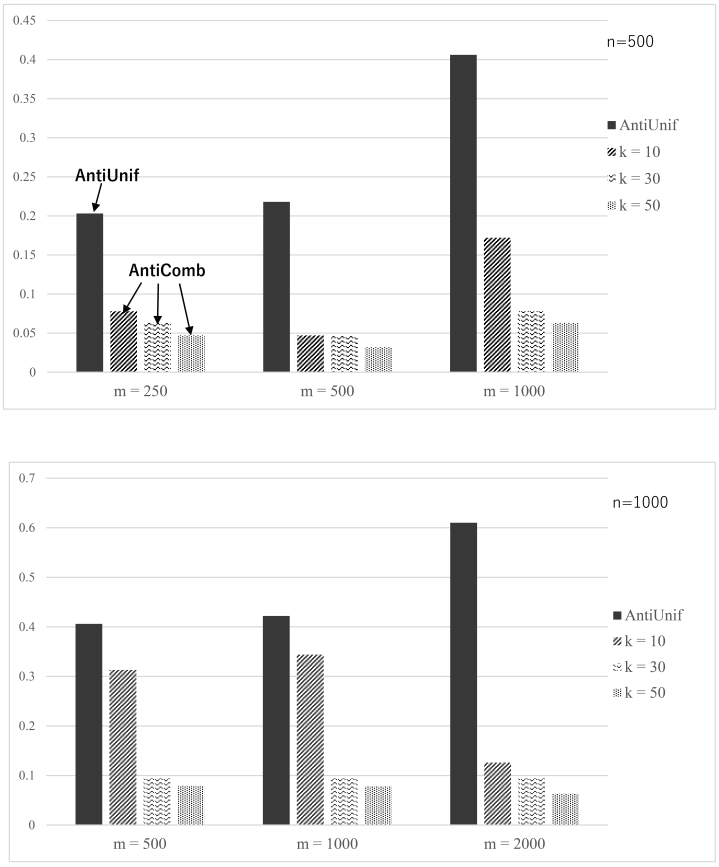


Fig. 4: Comparison of runtime by **AntiUnif** and **AntiComb** (1)

Several algorithms for anti-unification are proposed in the literature. Kuper $et\,al.$ [8] show that anti-unification of two terms represented in the form of trees of size n is carried out in time $O(\log^2 n)$ using n processors (or $n/\log^2 n$ processors in [2]). Kostylev and Zakharov [7] represent two given terms by acyclic directed graphs, and compute their most specific term (or lg) in time $O(N\log N)$ where N is the size of the most specific term it computes. In contrast to the algorithm proposed in this paper, those algorithms compute least generalization of two atoms (or terms). We use the Plotkin/Reynolds's algorithm in this paper, but $\mathbf{AntiComb}$ can use any algorithm of anti-unification of two atoms in the step 2. Kuper $et\,al.$ [8] also analyze that anti-unification of m terms, each having at most $O(n)$ symbols, is computed in $O(\log mn + \log^2 n)$ using mn processors. If we use their anti-unification algorithm of two atoms in $\mathbf{AntiComb}$, anti-unification of m atoms takes $O(m \times \log^2 n)$ using n processors. Using mn processors, it is done in $O(\log^2 n)$. Hence, $\mathbf{AntiComb}$ will be faster than anti-unification of m terms in [8].

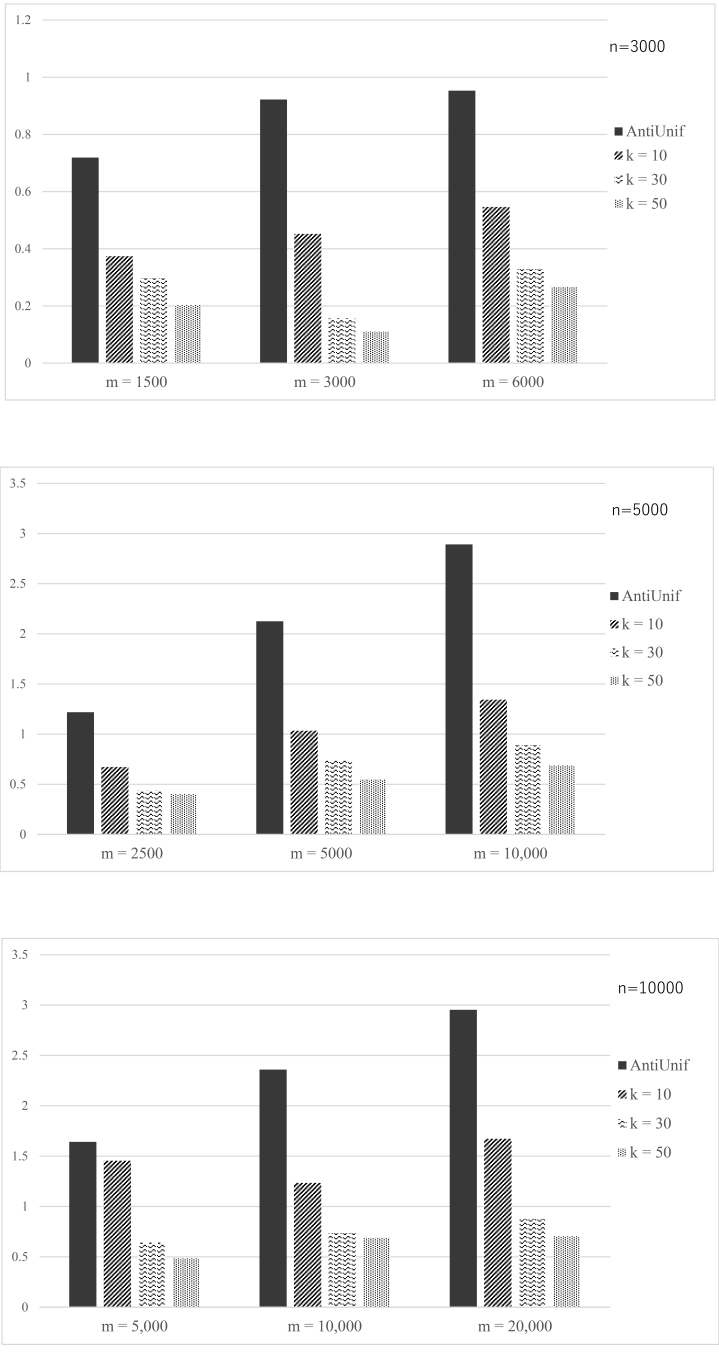


Fig. 5: Comparison of runtime by **AntiUnif** and **AntiComb** (2)

6 Conclusion

This paper introduced a new algorithm for computing a least generalization of a set of atoms based on anti-combination. Experimental results show that the proposed algorithm has potential to compute induction from big data in the form of relational facts in parallel. Future study includes extending the framework to generalization of clauses and exploiting other opportunities for parallelisation in practical ILP applications.

Acknowledgment
This research is funded by Vietnam National University – Ho Chi Minh city (VNU-HCM) under grant number C2019-26-01. We thank Mikio Yoshida for useful discussion on the subject of this paper.

References

1. Chang, C. L., Lee, R. T. C.: Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York (1973)
2. Delcher, A. L., Kasif, S.: Efficient parallel term matching and anti-unification. J. Automated Reasoning 9, 391–406 (1992)
3. N-Cheng, S-H., De Wolf, R. Foundations of Inductive Logic Programming, LNAI, vol. 1228, Springer, Berlin, Heidelberg (1997)
4. Eder, E.: Properties of substitutions and unifications. J. Symbolic Compt. 1, 31–46 (1985)
5. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to parallel computing (2nd edition), Addison Wesley (2003)
6. Gustafson, J. L.: Brent's Theorem. Encyclopedia of Parallel Computing, pp. 182–185. doi:10.1007/978-0-387-09766-4_80 (2011)
7. Kostylev, E. V., Zakharov, V. A.: On the complexity of the anti-unification problem. Discrete Mathematics Application 18(1), 85–98 (2008)
8. Kuper, G. M., McAloon,K. W., Palem, K. V., Perry, K., J.: A note on the parallel complexity of anti-unification. J. Automated Reasoning 9, 381–389 (1992)
9. Lassez, J.-L., Maher, M. J., Marriott, K.: Unification revisited. In: J. Mikner (ed.), Foundations of Deductive Databases and Logic Programming, pp. 587–625, Morgan Kaufmann (1988)
10. Lloyd, J. W.: Foundations of Logic Programming (2nd edition), Springer (1987)
11. Oancea, C., So, C, Watt, S., M.: Generalization in Maple. Maple Conference, pp. 377–382 (2005)
12. Østvold, B. M.: A functional reconstruction of anti-unification. Technical Report DART/04/04, Norwegian Computing (2004)
13. Palamidessi, C.: Algebraic properties of idempotent substitutions. In: Paterson M.S. (eds) Automata, Languages and Programming. ICALP 1990. LNCS, vol. 443, Springer, Berlin, Heidelberg (1990)
14. Plotkin, G. D.: A note on inductive generalization. Machine Intelligence, vol. 5, Edinburgh University Press, pp. 153–163 (1970)
15. Prawitz, D.: Advances and problems in mechanical proof procedures. Machine Intelligence, vol. 4, Edinburgh Univ. Press, pp. 59–71 (1969)
16. Reynolds, J. C.: Transformational systems and the algebraic structure of atomic formulas. Machine Intelligence, vol. 5, Edinburgh Univ. Press, pp. 135–151 (1970)
17. Yamasaki, S., Yoshida, M., Doshita, S.: A fixpoint semantics of Horn sentences based on substitution sets. Theoretical Computer Science 51, 309–324 (1986)
18. Yoshida, M. and Sakama, C.: Computing least generalization by anti-combination (short paper). Presented at ILP-14 (formally unpublished) (2014)