Computing Least Generalization by Anti-combination

Mikio Yoshida[1] and Chiaki Sakama[2]

[1] BBR Inc.
2-1-4-206, Sonezakishinchi, Kita-ku, Osaka, 530-0002, Japan
yos@bbr.jp
[2] Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp

Abstract. This paper provides a new method of computing a least generalization of terms. Using the technique of combination of substitutions, we introduce the notion of *anti-combination* which is the inverse substitution of combination. We then show that anti-combination is used for producing a least generalization of terms.

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 $P \cup \{G\}$ [5]. 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. Yamasaki and Yoshida [10, 11] introduce new resolution methods based on *combination* of mgus. The methods have the unique feature that resolution is performed by manipulation of substitutions, and mgus used in combination are computed independently of one another. Compared with composition of substitutions, however, combination of substitutions is of relatively little use in automated reasoning and logic programming. The notion is introduced by Prawitz [8] and is studied by Chang and Lee [1]. Mathematically, combination is obtained as the greatest lower bound of mgus [3].

In this paper we show that the combination operation is also used for computing *least generalization* (lg) of a set of terms. Plotkin [7] and Reynolds [9] introduce algorithms for *anti-unification* of two terms. Plotkin argue that it is iteratively applied to computing lg of a set of terms: for a set of terms $\{t_1, \ldots, t_n\}$, its lg is computed as $lg(t_1, lg(t_2, \ldots, lg(t_{n-1}, t_n) \cdots))$ where $lg(t_i, t_j)$ computes lg of t_i and t_j. We show that lg of a set of terms is computed by an inverse substitution of combination, which we call *anti-combination.* In Section 2 we review basic notions and algebraic properties of substitutions. In Section 3 we introduce a method of computing least generalization by anti-combination.

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, 5]. Variables are represented by letters $x, y, z, \ldots$; constants are represented by letters $a, b, c, \ldots$; and function symbols (of arities > 0) are represented by letters $f, g, h, \ldots$ A *term* is either (i) a constant, (ii) a variable, or (iii) the expression $f(t_1, \ldots, t_n)$ where f is an n-ary ($n > 0$) function symbol and $t_1, \ldots, t_n$ are terms. Two terms t_1 and t_2 are *compatible* if $t_1 = f(t_1, \ldots, t_k)$ for some f then $t_2 = f(t'_1, \ldots, t'_k)$. The set of all variables (resp. terms) in the language is denoted by Var (resp. $Term$). The set of variables occurring in a term t is denoted by $\mathcal{V}(t)$. Given a set $E = \{t_1, \ldots, t_n\}$ with $t_i \in Term$ $(1 \le i \le n)$, we assume $\mathcal{V}(t_i) \cap \mathcal{V}(t_j) = \emptyset$ for any $t_i \ne t_j$ $(1 \le i, j \le n)$. The following definitions of substitutions and their properties are due to [1, 3, 4, 11].

Definition 2.1 (substitution). A *substitution* is a mapping σ from Var into $Term$ such that the set $\Sigma = \{\langle x, \sigma(x)\rangle \mid x \ne \sigma(x) \text{ and } x \in Var\}$ 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 $\mathcal{D}(\sigma) = \{x \mid \langle x, t\rangle \in \Sigma\}$ is the *domain* of σ. The identity mapping ε over Var is the *empty substitution.* A bijection from Var to Var is a *renaming* of variables. The set of all renamings is denoted by Ren (where $Ren \subset Sub$).

Definition 2.2 (instance, gci). For $\sigma \in Sub$ and $t \in Term$, an *instance* $t\sigma$ of t is defined as follows:

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

For a set $E = \{t_1, \ldots, t_n\}$ with $t_i \in Term$ $(1 \le i \le n)$, a term t is a *greatest common instance* (gci) of E if (i) for any $t_i \in E$ there is $\sigma_i \in Sub$ such that $t = t_i\sigma_i$, and (ii) for any $s \in Term$ satisfying (i) it holds that $s = t\lambda$ for some $\lambda \in Sub$.

Definition 2.3 (composition, idempotent). For $\sigma, \lambda \in Sub$, the *composition* of σ and λ (denoted by $\sigma\lambda$) is a function from Var to $Term$ such that

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

A substitution σ is *idempotent* if $\sigma\sigma = \sigma$. The set of all idempotent substitutions is denoted by $ISub$. For any $t \in Term$, $t(\sigma\lambda) = (t\sigma)\lambda$. Also for any $\sigma, \lambda, \mu \in Sub$, it holds that $(\sigma\lambda)\mu = \sigma(\lambda\mu)$ and $\sigma\varepsilon = \varepsilon\sigma = \sigma$.

Definition 2.4 (unifier, mgu, mgsu). A substitution $\sigma \in Sub$ is a *unifier* for a set $E = \{t_1, \ldots, t_n\}$ with $t_i \in Term$ $(1 \le i \le n)$ if $t_1\sigma = \cdots = t_n\sigma$ holds. A

[3] It is often written as $\{x_1/t_1, ..., x_n/t_n\}$ [2, 5].

unifier σ for a set E is a *most general unifier* (mgu) (written $\sigma = mgu(E)$) if for each unifier θ for the set E, there is a substitution $\lambda \in Sub$ such that $\theta = \sigma\lambda$. For a finite set $\mathbf{E}$ of finite sets of terms, $\sigma \in Sub$ is a *most general simultaneous unifier* (mgsu) of $\mathbf{E}$ (written $mgsu(\mathbf{E})$) if $\sigma = mgu(E)$ for any $E \in \mathbf{E}$.

Definition 2.5 (order on $Term$). For any $s, t \in Term$, $s \leq t$ if $s = t\theta$ for some $\theta \in Sub$. In particular, we write $s \sim t$ if $s \leq t$ and $t \leq s$.

The relation $\leq$ is reflexive and transitive over the set $Term$. It holds that $s \sim t$ iff $s = t\lambda$ for some $\lambda \in Ren$. Let $\mathcal{T}$ be the quotient set $Term/\sim$, completed with the bottom element. Then the ordered set $(\mathcal{T}, \leq)$ constitutes a complete lattice [9].

Definition 2.6 (order on Sub). For any $\sigma, \theta \in Sub$, $\sigma \leq \theta$ if $\sigma = \theta\lambda$ for some $\lambda \in Sub$. In particular, we write $\sigma \sim \theta$ if $\sigma \leq \theta$ and $\theta \leq \sigma$.

The relation $\leq$ is reflexive and transitive over the set Sub.[4] It holds that $\sigma \sim \theta$ iff $\sigma = \theta\lambda$ for some $\lambda \in Ren$.

Proposition 2.1. [3, Prop.4.5] *For any finite set $E \subseteq Term$, $\sigma = mgu(E)$ for some $\sigma \in Sub$ iff there is $\lambda \in ISub$ such that $\lambda = mgu(E)$ and $\lambda \sim \sigma$.*

By Proposition 2.1 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. Then the ordered set $(\mathcal{IS}, \leq)$ constitutes a complete lattice [3].

Definition 2.7 (combination). For $\Theta \subseteq ISub$, the glb of $(\Theta, \leq)$ is called a *combination*. When $\Theta = \{\theta_1, ..., \theta_n\}$, it is written as $\theta_1 + \cdots + \theta_n$.

The operation '+' is commutative and associative. For any $\sigma \in ISub$, $\sigma + \sigma \sim \sigma$ and $\sigma + \varepsilon \sim \sigma$. Chang and Lee [1] provide another definition of combination. Given $\theta_1, ..., \theta_n \in ISub$ where $\theta_i = \{t^i_1/x^i_1, ..., t^i_{k_i}/x^i_{k_i}\}$ $(1 \leq i \leq n)$, the combination $\theta_1 + \cdots + \theta_n$ is defined as the mgu of two terms: $E_1 = f(t^1_1, \ldots, t^1_{k_1}, \ldots, t^n_1, \ldots, t^n_{k_n})$ and $E_2 = f(x^1_1, \ldots, x^1_{k_1}, \ldots, x^n_1, \ldots, x^n_{k_n})$ where f is a function symbol. These two definitions are proved to be equivalent [11].

Proposition 2.2. *For any $\sigma, \theta \in ISub$, if $\mathcal{D}(\sigma) \cap \mathcal{D}(\theta) = \emptyset$, $\sigma\theta = \sigma \cup \theta = \sigma + \theta$.*

Proposition 2.3. [3, 11] *Let $\mathbf{E} = \{E_1, \ldots, E_n\}$ be a set of finite sets of terms. (i) $mgsu(\mathbf{E}) \sim \sigma_1 \cdots \sigma_n$ where $\sigma_1 = mgu(E_1)$, $\ldots$, $\sigma_n = mgu(E_n\sigma_1 \cdots \sigma_{n-1})$, and (ii) $mgsu(\mathbf{E}) \sim mgu(E_1) + \cdots + mgu(E_n)$.*

Proposition 2.3 presents two different ways of computing $mgsu(\mathbf{E})$. The one (i) presents that computing $\sigma_1, \ldots, \sigma_n$ in a sequential manner and composing them to get $mgsu(\mathbf{E})$. This method is usually employed in binary resolution. The other one (ii) presents that computing $mgu(E_i)$ for each E_i and combining them to get $mgsu(\mathbf{E})$. Comparing two methods, computation of σ_i uses the results of $\sigma_1, \ldots, \sigma_{i-1}$ in (i). By contrast, in (ii) each $mgu(E_i)$ is computed independently, so that combination has potential usage in *parallel* computation of gci.

[4] We use the same symbol $\leq$ as the relation over $Term$, but the meaning is clear from the context.

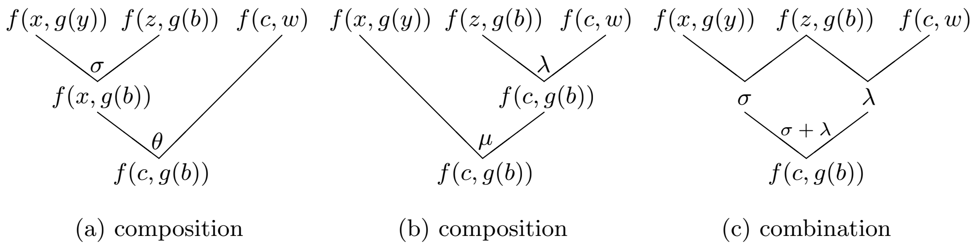


Fig. 1: composition and combination

Example 2.1. Let $E_1 = \{\, f(x,g(y)),\ f(z,g(b)) \,\}$ and $E_2 = \{\, f(z,g(b)),\ f(c,w) \,\}$. Then $\sigma = mgu(E_1) = \{b/y, x/z\}$ and $\theta = mgu(E_2\sigma) = \{c/x, g(b)/w\}$. The mgsu of $\{E_1, E_2\}$ is then obtained by the composition $\sigma\theta = \{c/x, b/y, c/z, g(b)/w\}$, and the gci of $E_1 \cup E_2$ is $f(c,g(b))$ (Fig. 1(a)). Similar computation is done by first computing $\lambda = mgu(E_2) = \{c/z, g(b)/w\}$ and then computing $\mu = mgu(E_1\lambda) = \{c/x, b/y\}$, which produces the same mgsu and the gci (Fig. 1(b)). On the other hand, the mgsu is computed by the combination $\sigma + \lambda = \{c/x, b/y, c/z, g(b)/w\}$ which produces the gci (Fig. 1(c)).

3 Computing Least Generalization by Anti-combination

For a set $E \subseteq Term$, its *least generalization* (lg) (written $lg(E)$) is defined as the least upper bound of the set $(E, \leq)$ [2]. We first provide the *anti-unification algorithm* [2, Algorithm 13.1] for computing a least generalization of two terms which is originally introduced in [9]. For two terms t_1 and t_2, let $s = lg(\{t_1, t_2\})$. Then there is a substitution $\theta_i \in ISub$ such that $t_i = s\theta_i$ $(i = 1, 2)$. We call θ_i *a substitution for lg* of $\{t_1, t_2\}$ (wrt t_i) and write $slg(t_i, \{t_1, t_2\}) = \theta_i$.

Definition 3.1 (anti-unification algorithm). [2, 9]
Input: two compatible terms t_1 and t_2
Output: $s = lg(\{t_1, t_2\})$ and $\theta_i = slg(t_i, \{t_1, t_2\})$ $(i = 1, 2)$

1. Set $t'_1 = t_1$ and $t'_2 = t_2$, $\theta_1 = \theta_2 = \varepsilon$, and $i = 0$.
   Let $z_1, z_2, \ldots$ be a sequence of variables not appearing in t_1 or t_2.
2. If $t'_1 = t'_2$, then output t'_1, θ_1, θ_2 and stop.
3. Let p be the leftmost symbol position where t'_1 and t'_2 differ. Let s_1 and s_2 be the terms occurring at this position in t'_1 and t'_2, respectively.
4. If, for some j with $1 \leq j \leq i$, $z_j\theta_1 = s_1$ and $z_j\theta_2 = s_2$, then replace s_1 at the position p in t'_1 by z_j, replace s_2 at the position p in t'_2 by z_j, and go to 2.
5. Otherwise set i to $i + 1$, replace s_1 at the position p in t'_1 by z_i, and replace s_2 at the position p in t'_2 by z_i. Set θ_1 to $\theta_1 \cup \{s_1/z_i\}$, θ_2 to $\theta_2 \cup \{s_2/z_i\}$, and go to 2.

Since the lub of $(E, \leq)$ is associative, the above anti-unification algorithm is iteratively applied for computing a least generalization of a set E of terms. In this case, each $slg(t_i, E)$ $(t_i \in E)$ is computed by a composition of substitutions.

Example 3.1. Let $E = \{t_1, t_2, t_3\}$, $s_1 = lg(\{t_1, t_2\})$ and $s_2 = lg(\{t_1, t_2, t_3\}) = lg(\{s_1, t_3\})$. Then $t_1 = s_1\theta_1$, $t_2 = s_1\theta_2$, $s_1 = s_2\sigma_1$, and $t_3 = s_2\sigma_2$ for some θ_1, θ_2, σ_1, $\sigma_2 \in Sub$. Then $t_1 = s_2\sigma_1\theta_1$, $t_2 = s_2\sigma_1\theta_2$, and $t_3 = s_2\sigma_2$. So $slg(t_1, E) = \sigma_1\theta_1$, $slg(t_2, E) = \sigma_1\theta_2$, and $slg(t_3, E) = \sigma_2\varepsilon$.

The above algorithm computes a substitution θ_i such that $t_i = s\theta_i$ $(i = 1, 2)$ for $s = lg(\{t_1, t_2\})$. Then an lg s is also computed by $s = t_i\theta_i^{-1}$ where θ_i^{-1} is an *inverse substitution* of θ_i. An inverse substitution θ^{-1} is well-defined if θ is injective.

Definition 3.2 (inverse substitution). [6] Let $\theta \in Sub$ be injective and $t \in Term$. If $\mathcal{V}(\mathcal{D}(\theta)) \cap \mathcal{V}(t) = \emptyset$, then an *inverse substitution* $\theta^{-1} : Term \to Var$ is defined as follows.

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

If $\mathcal{V}(\mathcal{D}(\theta)) \cap \mathcal{V}(t) \neq \emptyset$, a renaming substitution $\lambda \in Ren$ is applied to t in such a way that $\mathcal{V}(\mathcal{D}(\theta)) \cap \mathcal{V}(t\lambda) = \emptyset$. Then we can apply θ^{-1} to $t\lambda$ if θ is injective. If a substitution θ is not injective, we use the technique of [2] to constitute θ^{-1}. We do not have much space to present details and just give an example. 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.

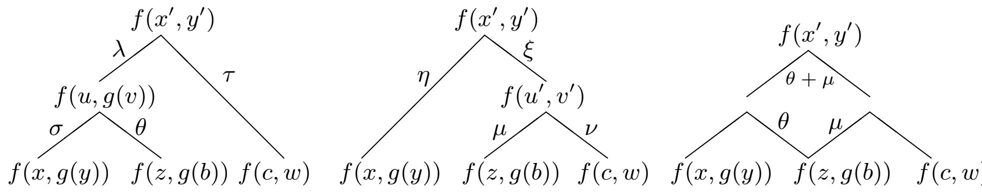
Definition 3.3 (anti-combination). Let $\sigma = \theta_1 + \cdots + \theta_n$ be a combination of $\{\theta_1, \ldots, \theta_n\} \subseteq ISub$. Then the inverse substitution σ^{-1} is called an *anti-combination* of $\theta_1, \ldots, \theta_n$.

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 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.

Proposition 3.1. *Let $E = \{t_1, t_2, t_3\}$ be a set of terms, $\theta = slg(t_2, \{t_1, t_2\})$ and $\mu = slg(t_2, \{t_2, t_3\})$. Then $lg(E) = t_2\sigma^{-1}$ where $\sigma \sim (\theta + \mu)$.*

Proof. Let $s_1 = lg(\{t_1, t_2\})$ and $s_2 = lg(\{t_2, t_3\})$. Then $lg(E) = lg(\{s_1, s_2\})$, and $s_1\theta = t_2$ and $s_2\mu = t_2$. By $\mathcal{D}(\theta) \cap \mathcal{D}(\mu) = \emptyset$, $s_1\theta = s_1(\theta + \mu) = t_2$ and $s_2\mu = s_2(\theta + \mu) = t_2$. Then $lg(E) = lg(\{s_1, s_2\}) = lg(\{t_2(\theta + \mu)^{-1}, t_2(\theta + \mu)^{-1}\}) = t_2(\theta + \mu)^{-1}$. Hence, the result holds. □

Since combination is associative, the result of Proposition 3.1 is extended to a set of terms containing n-terms $(n \geq 3)$.



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

Fig. 2: anti-unification and anti-combination

Example 3.2. Suppose the set $E = \{ f(x, g(y)),\ f(z, g(b)),\ f(c, w) \}$ of terms. Then $lg(\{ f(x, g(y)),\ f(z, g(b)) \}) = f(u, g(v))$ with $\sigma = \{x/u, y/v\}$ and $\theta = \{z/u, b/v\}$ where $f(u, g(v))\sigma = f(x, g(y))$ and $f(u, g(v))\theta = f(z, g(b))$. Next $lg(\{f(u, g(v)), f(c, w)\}) = f(x', y')$ with $\lambda = \{u/x', g(v)/y'\}$ and $\tau = \{c/x', w/y'\}$ where $f(x', y')\lambda = f(u, g(v))$ and $f(x', y')\tau = f(c, w)$. Then, $lg(E) = f(x', y')$ where $f(x', y')\lambda\sigma = f(x, g(y))$ with $\lambda\sigma = \{x/x', g(y)/y'\}$ and $f(x', y')\lambda\theta = f(z, g(b))$ with $\lambda\theta = \{z/x', g(b)/y'\}$ (Fig. 2(a)). Similar computation is done by first computing an lg of two terms $f(z, g(b))$ and $f(c, w)$ (Fig. 2(b)).

On the other hand, $\theta + \mu = \{z/u, b/v, z/u', g(b)/v'\}$. Then $(\theta + \mu)^{-1} = \{ (u/z, \langle \theta \rangle), (v/b, \langle \theta \rangle), (u'/z, \langle \mu \rangle), (v'/g(b), \langle \mu \rangle) \}$. Applying it to $f(z, g(b))$, $lg(E) = f(u, v')\ (\sim f(x', y'))$ is obtained (Fig. 2(c)). (Note: by the second condition of Def. 3.2, $(v/b, \langle \theta \rangle)$ is not applied to b in $f(z, g(b))$.)

References

1. C. L. Chang and R. T. C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.
2. S-H. N-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming. Lecture Notes in AI*, vol. 1228, Springer, 1997.
3. E. Eder. Properties of substitutions and unifications. *J. Symbolic Computation* 1:31–46, 1985.
4. J.-L. Lassez, M. J. Maher and K. Marriott. Unification revisited. In: J. Mikner (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 587–625, Morgan Kaufmann, 1988.
5. J. W. Lloyd. *Foundations of Logic Programming*, Springer, 1987.
6. B. M. Østvold. A functional reconstruction of anti-unification. In: *Technical Report DART/04/04, Norwegian Computing*, 2004.
7. G. D. Plotkin. A note on inductive generalization. In: *Machine Intelligence*, vol. 5, Edinburgh University Press, pp. 153–163, 1970.
8. D. Prawitz. Advances and problems in mechanical proof procedures. In: *Machine Intelligence*, vol. 4, Edinburgh Univ. Press, pp. 59–71, 1969.
9. J. C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In: *Machine Intelligence*, vol. 5, Edinburgh Univ. Press, pp. 135–151, 1970.
10. S. Yamasaki, M. Yoshida and S. Doshita. A fixpoint semantics of Horn sentences based on substitution sets. *Theoretical Computer Science* 51:309–324, 1986.
11. M. Yoshida, S. Yamasaki and S. Doshita. Formalization of unifiers separated resolution. *IEICE Trans. D-I*, Vol J72-D-I, No.6, pp. 455–464, 1989 (in Japanese).