Simple β_0-Unification for Terms with Context Holes

Makoto Hamana

Department of Computer Science, University of Gunma, Japan
hamana@cs.gunma-u.ac.jp

1 Introduction

Contexts are widely used notion in the lambda calculus and its application to computer science. Contexts are terms with some holes to be filled, and the difference between hole-filling and usual substitution is that filling a hole with a term can *capture* variable in the term by bound variable when the system has binding operators.

In this note we present a unification algorithm for terms involving both usual variables and holes. This algorithm is slightly simplified version of the one presented in the paper [Ham01]. An example of what we want to do is the following: consider the unification problem between lambda terms

$$\lambda x.M =^? \lambda x.f(x)$$

where M is a free variable. An intuitive answer $M \mapsto f(x)$ is not a unifier of the usual higher-order unification because of the properties of substitution avoiding capture of bound variable.

But we want to take this answer $M \mapsto f(x)$ as a unifier of the problem. If we regard M as *a hole of a context* filling by $f(x)$, capturing x by the binder is no problem. We consider this kind of a variant of higher-order unification between terms with holes modulo Miller's β_0-axiom [Mil91]. The β_0-axiom is defined by $(\lambda x.M)y = M[x := y]$, i.e. only an application with a variable causes substitution. This axiom is the basis of pattern unification [Mil91,Nip93].

The main benefit of our algorithm is decidability and simplicity. The unification problems we consider have variable (and hole) binding, whereas the algorithm is very similar to the first-order unification. The imitation and projection rules used in higher-order unification is no longer needed, the same effect can be done by a more simple rule taking just *inverse* of an intermediate substitution in a unification step.

The unification algorithm presented here have been used for a logic programming with variable binding [Ham01]. It is also hopeful to be a unification in the *calculi for contexts* [Mas99,HO01,SSK01] considered extensively in recent years, which are the lambda calculi extended with hole binding operator.

We should notice that the problem we consider is not *context unification* [Com92,SS94]. Context unification is a unification for terms with context variables (not holes) which are second-order variables only substituted by lambda terms with exactly one occurrence of the bound variable (meaning a hole), where holes themselves are not aimed to be unified.

2 Term Language

We will work with own term language to express the system having variable/hole-binding precisely. Terms are defined by

$$t ::= f(t_1, \ldots, t_n) \mid t@a \mid [a]t \mid \xi x \mid \mathsf{nam}(a)$$
$$\xi ::= [a_1 := a'_1, \ldots, a_n := a'_n]$$

where f is a function symbol, a's are *names* which are used for bindings. A term $t@a$ denotes an application, $[a]t$ an abstraction, $\mathsf{nam}(a)$ a name, and ξx a *variable with renamer* where ξ is an *renamer*. Types are defined by $\sigma ::= \iota \mid [\mathsf{N}]\sigma$ where ι is a base type and $[\mathsf{N}]\sigma$ is a type abstracted by a name. We use additional information of types, called stages, to control bound and free names occurring in a term. A type judgment is of the form $\Gamma \vdash t : \sigma : A$ where a typing context Γ is a finite set of variable declarations $\{x_1 : \sigma_1 : A_1, \ldots, x_n : \sigma_n : A_n\}$, σ is a *type*, and a *stage* A is a finite set of names expressing a possible free names occurring in a term. Well-typed terms are defined by the typing rules below.

$$\frac{A = \{\vec{a}\} \text{ and } B \supseteq \{\vec{b}\}}{\Gamma_1, x : \sigma : A, \Gamma_2 \vdash [\vec{a} := \vec{b}]x : \sigma : B} \qquad \frac{a \in A}{\Gamma \vdash \mathsf{nam}(a) : \iota : A}$$

$$\frac{\Gamma \vdash t_1 : \sigma_1 : A, \ldots, \Gamma \vdash t_n : \sigma_n : A}{\Gamma \vdash f(t_1, \ldots, t_n) : \sigma : A}$$

$$\frac{\Gamma \vdash t : \sigma : A}{\Gamma \vdash [a]t : [\mathsf{N}]\sigma : A - \{a\}} \qquad \frac{\Gamma \vdash t : [\mathsf{N}]\sigma : A}{\Gamma \vdash t@a : \sigma : A \cup \{a\}}\ a \notin \mathsf{BN}(t)$$

A *renaming* $\rho : A \to B$ is a function between sets of names presented by the notation $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$. A renaming ρ is extended to functions $\rho^\sharp$ and $\rho^\natural$ on terms from $T(\Gamma)$ to $T(\Gamma)$.

3 Unification

Next, we consider unification of terms modulo the equational theory generated by the following axioms:

$$\begin{aligned}
(\alpha) &\quad \Gamma \vdash [a]t = [a'][a \mapsto a']^\sharp t : [\mathsf{N}]\sigma : A \\
(\beta_0) &\quad \Gamma \vdash ([a]t)@b = [a \mapsto b]^\sharp t : \sigma : A \\
(\eta) &\quad \Gamma \vdash [a](t@a) = t : [\mathsf{N}]\sigma : A.
\end{aligned}$$

and inference rules:

Rules for congruence

$$(\mathrm{Ax})\frac{\Gamma \vdash s = t : \sigma : A \in \{(\alpha), (\beta_0), (\eta)\} \cup E}{\Gamma \vdash s = t : \sigma : A}$$

$$(\text{Congr-}f)\frac{\Gamma \vdash t_i = t'_i : \sigma_i : A \quad f : \sigma_1, \ldots, \sigma_n \to \sigma \in \Sigma}{\Gamma \vdash f(t_1, \ldots, t_i, \ldots, t_n) = f(t_1, \ldots, t'_i, \ldots, t_n) : \sigma : A}$$

$$(\text{Congr-}[])\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash [a]s = [a]t : [\mathsf{N}]\sigma : A - \{a\}}\ a \in A$$

$$(\text{Congr-}@)\frac{\Gamma \vdash s = t : [\mathsf{N}]\sigma : A}{\Gamma \vdash s@a = t@a : \sigma : A \cup \{a\}}\ a \notin A$$

$$(\mathrm{Subst})\frac{\Gamma \vdash s : \sigma : A \quad \Gamma, x : \sigma : A \vdash t = t' : \sigma : B}{\Gamma \vdash [x \mapsto s]t = [x \mapsto s]t' : \sigma : B}$$

Rules for equivalence relation

$$(\mathrm{Ref})\frac{\Gamma \vdash t : \sigma : A}{\Gamma \vdash t = t : \sigma : A} \qquad (\mathrm{Sym})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash t = s : \sigma : A}$$

$$(\mathrm{Tr})\frac{\Gamma \vdash s = t : \sigma : A \quad \Gamma \vdash t = u : \sigma : A}{\Gamma \vdash s = u : \sigma : A}$$

Rules for variation

$$(\natural)\frac{\Gamma \vdash s = t : \sigma : A}{\rho\Gamma \vdash \rho^\natural s = \rho^\natural t : \sigma : \rho A}\ \rho \text{ is a 1-1 renaming}$$

Structural rules

$$(\mathrm{Wk})\frac{\Gamma \vdash s = t : \sigma : B}{\Gamma,\ x{:}\sigma{:}A \vdash s = t : \sigma : B} \qquad (\text{Wk-r})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash s = t : \sigma : A \cup \{a\}}\ a \notin \mathsf{BN}(s) \cup \mathsf{BN}(t)$$

$$(\text{Str-r})\frac{\Gamma \vdash s = t : \sigma : A}{\Gamma \vdash s = t : \sigma : A - \{a\}}\ a \notin \mathsf{FN}(s) \cup \mathsf{FN}(t)$$

Here, $\mathsf{FN}(t)$ and $\mathsf{BN}(t)$ denote sets of free and bound names respectively.

A *unification problem* is a pair $(\Gamma \vdash t : \tau : A,\ \Gamma' \vdash t' : \tau : A')$ of terms. A *unifier* to the unification problem is a pair (θ, ρ) satisfying $\Gamma'' \vdash \theta\rho^\natural t = \theta t' : \tau : A'$ where $\theta : \rho\Gamma \cup \Gamma' \to T(\Gamma'')$ is a substitution, ρ is a 1-1 renaming between the sets of names in the pair of terms.

A *standardized unification problem* is a unification problem satisfying the following properties: (i) The names in two terms are mutually disjoint. (ii) Γ and Γ' are mutually disjoint. (iii) $|A| = |A'|$. (iv) t and t' has no β_0, η-redexes. (v) The bound variables in t and t' are numbered by the method of de Bruijn level. (vi) The types of all variables are ι.

We can show that any unification problem can be a standardized unification problem, so it is sufficient to consider a unification algorithm for standardized unification problem. Also we can easily recover a unifier of the original problem from the unifier of standardized one. See detail [Ham01].

Unification algorithm BUA. The unification problem is extended to a pair of sequences of terms and a unifier is similarly defined. For a unification problem, a *system* is of the form $t_1{:}A_1 =^? t'_1{:}A'_1, \ldots, t_n{:}A_n =^? t'_n{:}A'_n$. The unification algorithm BUA on systems given in Fig. 1 computes unifiers for a given unification problem. This is based on Martelli and Montanari's unification algorithm for first-order terms by the set of transformations. In the rules (v-elim1) and

$$(f\text{-elim})\frac{G_1, f(s_1,\ldots,s_n) : A =^? f(t_1,\ldots,t_n) : B, G_2}{G_1, s_1 : A =^? t_1 : B, \ldots, s_n : A =^? t_n : B, G_2}$$

$$(\text{triv})\frac{G_1, \xi x : A =^? \xi x : A, G_2}{G_1, G_2} \qquad ([]\text{-elim})\frac{G_1, [k]s : A =^? [k]t : B, G_2}{G_1, s : A \cup \{k\} =^? t : B \cup \{k\}, G_2}$$

$$(\text{ov-elim})\frac{G_1, \mathsf{nam}(a) : A =^? \mathsf{nam}(b) : B, G_2}{\rho(G_1, s : A =^? t : B, G_2)} \qquad (@\text{-elim})\frac{G_1, s@a : A =^? t@b : B, G_2}{\rho(G_1, s : A =^? t : B, G_2)}$$

$$(\text{v-elim1})\frac{G_1, \xi x : B =^? t : A, G_2}{\theta\rho(G_1, G_2)} \qquad (\text{v-elim2})\frac{G_1, t : A =^? \xi x : B, G_2}{\theta\rho(G_1, G_2)}$$

In (v-elim1) and (v-elim2), $x \notin \mathsf{VAR}(t)$.

Fig. 1. Unification algorithm BUA

(v-elim2) of BUA, the following data is used:

- $x : \tau : D \in \Gamma$ ($\in \Gamma'$ in (v-elim2)),
- a bijective renaming $\rho : B \to A$, where $A = \langle a_1, \ldots, a_n\rangle$, $B = \langle b_1, \ldots, b_n\rangle$, is defined by $\rho(i) \triangleq i$.
- an renaming $\xi^\diamond : D \to B$,
- a substitution θ is arbitrarily taken from the set $\Theta = \{[x{:}\tau{:}D \mapsto s{:}\tau{:}D] \mid \langle s, E\rangle \in S,\ E \subseteq D\}$ (i.e. there are $\#\Theta$-different θ's) where

$$S \triangleq \{\langle t[p_1 := \xi'_1 \circ \rho^{-1}(t_{|p_1}), \ldots, p_n := \xi'_n \circ \rho^{-1}(t_{|p_n})], \bigcup_i \xi'_i B\rangle$$
$$\mid \{p_1, \ldots, p_n\} = \mathsf{FNPOS}(t) \text{ and } p_1 \neq \ldots \neq p_n,$$
$$\xi'_1, \ldots, \xi'_n \in \{\xi' : B \to D \mid \xi^\diamond \circ \xi' = \mathsf{id}_B\} \cup \{\mathsf{id}_B\}\}.$$

Note that $\mathsf{FNPOS}(t)$ denotes the set of all positions of free names occurring in t, $t[p := s]$ is a replacement of a term t at the position p with a term s, and $t_{\restriction p}$ denotes a subterm of t at the position p. Also (v-elim1) and (v-elim2) produce $\#\Theta$-different branches of derivation.

Theorem 1. *(*[**Ham01**]*) The algorithm BUA is sound, complete and decidable, and the unifiers are $\beta_0\eta$-normalized.*

Not all pattern unification problems can be solved by BUA because our term language only admits a restricted form of application (i.e. $([a]t)@b$, b is a name). However, the syntactic restriction of unification problems for BUA is imposed by just well-typedness, namely BUA can solve *non-pattern* unification problems if they are well-typed. The following is an example of such a problem and solving steps by BUA.

Example. Consider a non-pattern unification problem $\lambda a.Faa =^? \lambda d.gd$ where F is a free variable. The right-hand side is not a higher-order pattern because the bound variable d is applied twice to the free variable. Corresponding unification problem in our syntax is

$$(F{:}[\mathsf{N}][\mathsf{N}]\iota{:}\varnothing \vdash [a]F@a@a : [\mathsf{N}]\iota : \varnothing, \quad \vdash [d]g(\mathsf{nam}(d)) : [\mathsf{N}]\iota : \varnothing).$$

The first step is standardization. The higher-order variable F is replaced with $[b][c]F'$ using the first-order variable F' and the outermost bound name is de Bruijn indexed: $[1]([b][c]F')@1@1 : \varnothing =^? [1](g(\mathsf{nam}(1))) : \varnothing$. β_0-normalization to the left-hand side gives

$$[1]([b := 1, c := 1]F') : \varnothing =^? [1](g(\mathsf{nam}(1))) : \varnothing.$$

Stripping the binder [1], we can apply (v-elim1)-rule:

$$([b := 1, c := 1]F') : \{1\} =^? (g(\mathsf{nam}(1))) : \{1\}$$
$$F' \mapsto g(\mathsf{nam}(b)), \quad F' \mapsto g(\mathsf{nam}(c)).$$

So the unifiers are $F \mapsto [b][c]g(\mathsf{nam}(b))$ and $F \mapsto [b][c]g(\mathsf{nam}(c))$.

In this (v-elim) step, these answers are obtained by applying all possible *inverse* substitutions computed in the set S of BUA (this case is $[1 \mapsto b]$ and $[1 \mapsto c]$ for $[b := 1, c := 1]$) to the right-hand side. This is the crucial point why BUA can solve non-pattern unification problem by very simple way.

References

[Com92] H. Comon. Completion of rewrite systems with membership constraints. In *Coll. on Automata, Languages and Programming, LNCS 623*, 1992.

[Ham01] M. Hamana. A logic programming language based on binding algebras. In N. Kobayashi and B. C. Pierce, editors, *4th International Symposium on Theoretical Aspects of Computer Software, TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 243–262. Springer, 2001.

[HO01] M. Hashimoto and A. Ohori. A typed context calculus. *Theoretical Computer Science*, 266:249–271, 2001.

[Mas99] I. Mason. Computing with contexts. *Higher-Order and Symbolic Computation 12*, pages 171–201, 1999.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science,*, pages 64–74, 1993.

[SS94] M. Schmidt-Schauß. Unification of stratified second-order terms. Technical Report 12/94, J. W. Goethe Universität, Frankfurt, 1994.

[SSK01] M. Sato, T. Sakurai, and Y. Kameyama. A simply typed context calculus with first-class environments. In *5th International Symposium on Functional and Logic Programming*, volume LNCS 2024, pages 359–374, 2001.