Generalization at Higher Types

Robert W. Hasker [1] and Uday S. Reddy
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
{hasker, reddy}@cs.uiuc.edu

1 Abstract

Finding the most specific generalization of first-order terms is well understood, but the generalization of higher-order terms remains unsolved. We provide a framework for the second-order case by using a categorical structure for terms and identifying a class of useful generalizations called *relevant* generalizations. We show that complete sets of maximally specific generalizations exist and are computable. Such generalizations have an important application for proof modification in automated proof systems.

2 Introduction

Automated proof development systems, including program verification systems, program construction systems, and program transformation systems [4, 10, 2, 15] face the problem of how to incorporate *modifications*. Having constructed a proof for a theorem (or, a program for a specification) as a combination of manual and automated effort, we would certainly not wish to redo the entire effort when the theorem is slightly modified. There is no great damage in redoing the automated part of the proof, but redoing the manual part of the proof manually could be too cumbersome. An ideal automated system should be able to compare the old and new theorems, keep track of the differences, and apply the steps of the old proof to the new theorem as long as they are applicable. We call such a system a *replay* system.

A fundamental problem in building replay systems is drawing analogies between the old and new theorems. The problem can be restated in terms of anti-unification [14, 16]. Given two terms t and u, find the most specific generalization g of the two terms together with the attendant substitutions $\theta : g \to t$ and $\sigma : g \to u$. The triple $\langle g, \theta, \sigma \rangle$, called the *anti-unifier* of t and u, contains the information necessary to relate the subterms of t and u.

If t and u are first-order terms, their first-order anti-unifier can be computed using well-known techniques [14, 16, 8]. However, in modern proof systems the formulas and terms involved are higher-order [7, 11, 10, 4]. Secondly, even if the terms are first-order, the first-order anti-unifier does not contain enough information to relate all corresponding subterms. For instance, if a formula A is replaced by a conjunction $\mathsf{A} \wedge \mathsf{B}$, the first-order anti-unifier gives the trivial generalization x, loosing the information that A appears in both the formulas. Another common modification often made is to add parameters to functions and predicates. However, the first-order anti-unifier of $f(t)$

[1] The work by Hasker was supported in part by a grant from Motorola Corp.

and $g(t, u)$ is again trivial. Thus, higher-order generalization is necessary to compute analogies in a replay system.

Surprisingly, even though the first-order anti-unification algorithms has been known since [14, 16], its higher-order counterpart does not seem to have received attention. Recently, [12] gave an algorithm for anti-unifiers for a special class of terms called *patterns* (terms restricted so that only abstraction variables can appear as arguments of a free variable), but the general case is yet unsolved. The pattern restriction precludes using such anti-unifiers in replay systems because it generates nearly the same anti-unifier as in the first-order case. In fact, the difficulties in generalizing higher-order terms while allowing for common subterms are considerable. While first-order terms form a complete lattice with unifiers as infs and anti-unifiers as sups, higher-order terms do not even posses infs. Huet's [8] "algorithm" computes a complete set of minimal unifiers, but the set can be infinite. For the opposite direction of upper bounds, we show that complete sets do not exist, in general. In fact, we believe that the naïve notion of "more general than" used in the first-order case is not meaningful for higher-order terms.

In this paper, we consider the problem of generalization restricted to second-order terms. We define the notion of generality using a categorical framework with substitutions as morphisms between terms. Complete sets of generalizations do not exist even in this setting, but we note that this is due to certain trivial generalizations. By restricting attention to nontrivial generalizations (called *relevant* generalizations), we find that complete sets exist and have interesting properties. We also show that the complete sets are computable and give a semi-practical algorithm for computing them.

3 Notation

We will generalize simply-typed λ-terms [3]. If $C = \uplus_\tau C_\tau$ is the set of constants and $V = \uplus_\tau V_\tau$ the variables, then a term is well-typed if it is consistent with the rules

$$\frac{c \in C_\tau}{c : \tau} \qquad\qquad \frac{x \in V_\tau}{x : \tau}$$

$$\frac{t : \tau \to \tau' \quad u : \tau}{t\, u : \tau'} \qquad\qquad \frac{x : \tau \quad t : \tau'}{\lambda x.t : \tau \to \tau'}$$

We use the convention that constants are set in `type` and variables in *italics.* We assume all terms are well-typed.

The order of a type τ is defined as

$$\begin{array}{lcl} \mathrm{order}(D_i) & = & 1 \\ \mathrm{order}(\tau \to \tau') & = & \max(1 + \mathrm{order}(\tau), \mathrm{order}(\tau')) \end{array}$$

The order of a term is just the order of its type. In this paper, we assume all terms are first or second-order, so all constants are in $C_{D_1 \to \ldots \to D_n}$ and all variables in $V_{D_1 \to \ldots \to D_m}$ for $n, m \geq 1$.

We assume the usual α, β, and η conversion rules. All equivalences between λ-terms and substitutions over λ-terms are assumed to be modulo λ-conversion. Application associates to the left and → to the right; parentheses are often dropped when they are not needed. By the Church-Rosser and strong normalization properties of typed λ-calculus (see, *eg*, [5]), every term of type $D_1 \rightarrow \ldots \rightarrow D_n \rightarrow D_{n+1}$ can be written in the form[2]

$$\lambda x_1.\lambda x_2.\cdots\lambda x_n.hu_1u_2\ldots u_m$$

where each x_i is distinct, $h \in C_\tau \cup V_\tau$, and each u_i is in normal form. We call h the *head* and $\overline{u}$ the *arguments*. Following [8], we say that a term is *flexible* if its head is a free variable and *rigid* otherwise (*ie*, if it is a constant or a bound variable). We will abbreviate terms in normal form as $\lambda x_1 \cdots x_n.h(u_1, \ldots, u_m)$ or even as $\lambda\overline{x_n}.h(\overline{u_m})$ where $\overline{x_n}$ denotes the sequence $x_1, \ldots, x_n$. If n is 0, then $\overline{x_n}$ is the empty sequence, and if n is arbitrary (but finite) we denote the sequence as just $\overline{x}$. The identity function $\lambda x.x$ is abbreviated as π and the general projection function $\lambda\overline{x_n}.x_k$ as π^n_k. The set of free variables in the term t is $\mathcal{FV}(t)$, and the set of bound variables is $\mathcal{BV}(t)$. The context of u in t is denoted $t[u]$ or alternatively as $t[\alpha \leftarrow u]$ if its position is relevant.

A substitution θ is a finite map from variables to type-equivalent terms with the domain denoted as $dom(\theta)$ and free variables in the range as $ran(\theta)$. θ_{id} denotes the identity substitution. Application of θ onto term t is variously denoted by $\theta(t)$ and $\theta : t \rightarrow u$ (where $u = \theta(t)$). The composition of two substitutions is defined as $\theta \circ \sigma = \lambda t.\theta(\sigma(t))$. To make substitutions easier to read, we will often leave the variables being bound implicit: if the substitution θ is being applied to term u, we may write it as $[\theta(x_1), \ldots, \theta(x_n)]$ where $\langle x_1, \ldots, x_n\rangle$ are the free variables listed in the order they occur when reading u from left to right.

4 The category of generalizations

First-order generalizations can be compared using the preorder $v \leq u \iff \exists\theta.\theta : v \rightarrow u$. This ordering is adequate because the substitution is unique, but in the higher-order case it often is not. Category theory provides a framework which supports distinguishing between substitutions.

In this section we examine the category of terms and show that it is inadequate. This leads to the category of generalizations and a discussion of its inadequacies.

Definition 4.1 Given a type τ, the category $\mathbf{T}_\tau$ has as objects terms of type τ. The arrows of $\mathbf{T}_\tau$ are given by substitutions $\theta : t \rightarrow u$ such that $dom(\theta) = \mathcal{FV}(t)$, $ran(\theta) = \mathcal{FV}(u)$, and $\theta(t) = u$. The composition is substitution composition and the identity arrows are identity substitutions.

We often leave the type subscript τ implicit. When $\theta : t \rightarrow u$ we say that t is *more general* than u (or conversely, u is *more specific* than t). But, θ indicates in what way t is more general than u. For first-order terms, $\mathbf{T}$ is a preorder; *ie*, there is at most one substitution between any two terms. For second-order terms, this is not the case; for example,

$$[\lambda x.g(x, \mathsf{a})] \text{ and } [\lambda x.g(x, x)] : f\mathsf{a} \rightarrow g(\mathsf{a}, \mathsf{a})$$

That is, for the second order case, a term may generalize another in multiple ways. This is the motivation for considering categories instead of preorders.

[2] Note that such forms are in normal form with respect to β, but not η.

Definition 4.2 If $a \in \mathbf{T}$ is a term, the category $\mathbf{G}(a)$—of *generalizations* of a—has as its objects substitutions $\theta : t \to a$ for $t \in \mathbf{T}$. A (generalization) morphism $\rho : (\theta_1 : t_1 \to a) \to (\theta_2 : t_2 \to a)$ is a substitution $\rho : t_1 \to t_2$ such that the following triangle commutes:

$$\begin{tikzcd}
t_2 \arrow[r, "\theta_2"] & a \\
t_1 \arrow[u, "\rho"'] \arrow[ur, "\theta_1"'] &
\end{tikzcd}$$

($\mathbf{G}(a)$ is often called the "slice category" $\mathbf{T}_\tau/a$.)[3]

This definition can be extended to generalizations of multiple terms. We show the binary case:

Definition 4.3 If $a_1, a_2 \in \mathbf{T}$, the category $\mathbf{G}(a_1, a_2)$ has as its objects pairs of substitutions $\langle \theta_1 : t \to a_1, \theta_2 : t \to a_2 \rangle$. A morphism

$$\rho : \langle \theta_1 : t \to a_1, \theta_2 : t \to a_2 \rangle \to \langle \sigma_1 : u \to a_1, \sigma_2 : u \to a_2 \rangle$$

is a substitution $\rho : t \to u$ that is a generalization morphism in both $\mathbf{G}(a_1)$ and $\mathbf{G}(a_2)$. That is, the following diagram commutes:

$$\begin{tikzcd}
a_1 & u \arrow[l, "\sigma_1"'] \arrow[r, "\sigma_2"] & a_2 \\
& t \arrow[ul, "\theta_1"] \arrow[u, "\rho"'] \arrow[ur, "\theta_2"'] &
\end{tikzcd}$$

As an aside, note that $\mathbf{G}(a_1, a_2)$ is the pullback $\mathbf{G}(a_1) \times_{\mathbf{T}} \mathbf{G}(a_2)$ in **Cat**. That is, if *src* is the forgetful functor, the diagram

$$\begin{tikzcd}
\mathbf{G}(a_1, a_2) \arrow[r, "\pi_2"] \arrow[d, "\pi_1"'] & \mathbf{G}(a_2) \arrow[d, "src"] \\
\mathbf{G}(a_1) \arrow[r, "src"] & \mathbf{T}_\tau
\end{tikzcd}$$

commutes.

[3] Note that $\mathbf{T}_\tau$ can itself be treated as a slice category $\mathbf{Type}/\tau$ where **Type** is the category of types [9, 6].

Examples 4.4 The following examples illustrate that generalizations which include common subterms are more specific:

$$\begin{tikzcd}
\mathsf{Dc} & h\mathsf{c} \arrow[l, "{[\mathsf{D}]}"'] \arrow[r, "{[\mathsf{E}]}"] & \mathsf{Ec} \\
& x \arrow[ul, "{[\mathsf{Dc}]}"] \arrow[u, "{[h\mathsf{c}]}"'] \arrow[ur, "{[\mathsf{Ec}]}"'] &
\end{tikzcd}$$

These generalizations are not isomorphic because the only substitution from $h\mathsf{c}$ to x, $\{h \mapsto \lambda y.x\}$, is not a generalization morphism. In comparison, [12] disallows the more specific of the two generalizations because $h\mathsf{c}$ is not a valid pattern. The only generalization meeting the pattern restriction is $\langle [\mathsf{Dc}] : x \mapsto \mathsf{Dc},\ [\mathsf{Ec}] : x \mapsto \mathsf{Ec} \rangle$, thus patterns do not capture common subterms.

$$\begin{tikzcd}
\lambda x.\mathsf{P}(x) \wedge \mathsf{Q}(x) & \lambda x.f(\mathsf{P}(x), \mathsf{Q}(x)) \arrow[l, "{[\wedge]}"'] \arrow[r, "{[\vee]}"] & \lambda x.\mathsf{P}(x) \vee \mathsf{Q}(x) \\
& \lambda x.g(\mathsf{P}(x), x) \arrow[ul, "{[\lambda yz.y \wedge \mathsf{Q}(z)]}"] \arrow[u, "{[\lambda yz.f(y, \mathsf{Q}(z))]}"'] \arrow[ur, "{[\lambda yz.y \vee \mathsf{Q}(z)]}"'] &
\end{tikzcd}$$

These are not isomorphic because there is no substitution from $\lambda x.f(\mathsf{P}(x), \mathsf{Q}(x))$ to $\lambda x.g(\mathsf{P}(x), x)$.

Examples 4.5 It is also instructive to examine generalizations which are unrelated by morphisms. The first illustrates that for two generalizations to be related, subterms must be used consistently:

$$[\lambda x.\mathsf{D}(x, \mathsf{b})] : f\mathsf{a} \to \mathsf{D}(\mathsf{a}, \mathsf{b}) \ \text{ and } \ [\lambda x.\mathsf{D}(\mathsf{a}, x)] : g\mathsf{b} \to \mathsf{D}(\mathsf{a}, \mathsf{b})$$

These are unrelated because any generalization morphism would have to eliminate the a (from $f\mathsf{a}$) or b (from $g\mathsf{b}$). The second example illustrates that different substitutions give rise to unique generalizations:

$$[\lambda xy.\mathsf{E}(x, x, y)] : \lambda z.f(z, z) \to \lambda z.\mathsf{E}(z, z, z)$$
$$[\lambda xy.\mathsf{E}(y, x, x)] : \lambda z.g(z, z) \to \lambda z.\mathsf{E}(z, z, z)$$

These are unrelated because the substitutions project distinct arguments.

Two generalizations g_1 and g_2 are *isomorphic*, written $g_1 \cong g_2$, if there are $\rho : g_1 \to g_2$ and $\rho_{op} : g_2 \to g_1$ such that $\rho \circ \rho_{op} = \theta_{id} = \rho_{op} \circ \rho$. We can show that isomorphisms are renamings.

Definition 4.6 ([12]) θ is a *renaming* iff for all $f \in dom(\theta)$, $\theta(f) = \lambda \overline{x}.h(\overline{x'})$ where h is a variable and $\overline{x'}$ is a permutation of $\overline{x}$.

Lemma 4.7 $g_1 \cong g_2$ by $\rho : g_1 \to g_2$ and $\rho_{op} : g_2 \to g_1$ iff ρ and ρ_{op} are renamings.

This follows from the observation that whenever $\theta(\sigma(f)) = f$ and $\sigma(f) = \lambda \overline{x}.t$, t must be flexible and all $x_i \in \overline{x}$ occur in t.

Observation 4.8 $([a] : f \to a)$ is initial in $\mathbf{G}(a)$.

This is because there is only one substitution between $\lambda\overline{x}.f(\overline{x})$ and any term. Since θ_{id} is a left identity,

Observation 4.9 $\theta_{id} : a \to a$ is terminal in $\mathbf{G}(a)$.

However, the morphisms of $\mathbf{G}(a)$ are not always unique:

Example 4.10

$$\begin{tikzcd}
\lambda x.h(x) \arrow[r, "{[\pi]}"] & \lambda x.x \\
\lambda x.f(g(x)) \arrow[u, "{[h,\pi],\text{ or } [\pi,h]}"] \arrow[ur, "{[\pi,\pi]}"'] &
\end{tikzcd}$$

Another difficulty is that $\mathbf{G}$ is not well-behaved with respect to maximal objects. Ideally, the maximally specific generalizations of any two terms a and b would be the maximal objects of $\mathbf{G}(a, b)$. However, the maximal objects are often undefined. The following examples show that the sources of maximal objects have unbounded depth and width. We also show that the arbitrarily large terms are not isomorphic to smaller terms, thus defining maximal objects up to an isomorphism would not be sufficient.

Example 4.11 Consider $\mathbf{G}(\mathsf{D}a, \mathsf{E}b)$, where a and b are arbitrary terms:

$$\begin{tikzcd}
\mathsf{D}a & f(g(a,b)) \arrow[l, "{[\mathsf{D},\pi_1^2]}"'] \arrow[r, "{[\pi_1^1,\mathsf{E}\circ\pi_2^2]}"] & \mathsf{E}b \\
& h(a,b) \arrow[ul, "{[\mathsf{D}\circ\pi_1^2]}"] \arrow[u, "{[f\circ g]}"] \arrow[ur, "{[\mathsf{E}\circ\pi_2^2]}"'] &
\end{tikzcd}$$

Note that the two generalizations are not isomorphic because there is no generalization morphism in the opposite direction. If ρ was such a morphism, then $\mathit{head}(\rho(f)) = h$ or $\rho(f) = \pi$, but neither choice allows both sides to commute simultaneously. Similarly, g can be mapped to $f' \circ g'$ and so on. A generalization morphism in the opposite direction can be found after a few repetitions of the pattern, but the generalizations remain nonisomorphic.

Example 4.12 $\mathbf{G}(\mathsf{c}, \mathsf{d})$ contains

$$\begin{tikzcd}
{[\mathsf{c}]} & g(\mathsf{c},\mathsf{d},\mathsf{e}) \arrow[l, "{[\pi_1^2]}"'] \arrow[r, "{[\pi_2^2]}"] \arrow[d, shift right, "{[\lambda xyz.f(x,y)]}"'] & \mathsf{d} \\
& f(\mathsf{c},\mathsf{d}) \arrow[ul, "{[\pi_1^2]}"] \arrow[ur, "{[\pi_2^2]}"'] \arrow[u, shift right, "{[\lambda xy.g(x,y,\mathsf{e})]}"'] &
\end{tikzcd}$$

Again, these are not isomorphic. This example can also be generalized to an arbitrary number of subterms in place of e. A similar situation occurs when bound variables are repeated arbitrarily often:

Example 4.13 $\mathbf{G}(\lambda x.\mathsf{D}x, \lambda x.\mathsf{E}x)$ contains

$$\begin{tikzcd}
\lambda x.\mathsf{D}x & \lambda x.g(x,x,x) \arrow[l, "{[\lambda uvw.\mathsf{D}u]}"'] \arrow[r, "{[\lambda uvw.\mathsf{E}w]}"] \arrow[d, shift right, "{[\lambda uvw.f(u,w)]}"'] & \lambda x.\mathsf{E}x \\
& \lambda x.f(x,x) \arrow[ul, "{[\lambda uv.\mathsf{D}u]}"] \arrow[ur, "{[\lambda uv.\mathsf{E}v]}"'] \arrow[u, shift right, "{[\lambda uv.g(u,u,v)]}"'] &
\end{tikzcd}$$

5 Relevant generalizations

These examples show that while $\mathbf{G}$ may be more a suitable category in which to find maximal generalizations than $\mathbf{T}$, it is not ideal. We can improve on $\mathbf{G}$ by restricting attention to only those generalizations which are *relevant*, where relevance means that each subterm is useful in forming the generalization. In particular, the following definitions permit variables only when they are necessary and permit rigid subterms only when they are actually used.

Example 4.11 suggests disallowing nested flexible subterms. We use the following definitions:

Definition 5.1 A generalization $\theta : t \to a$ is said to be *redundant* if t has a subterm of the form $f(\ldots, g(\ldots), \ldots)$ and $\theta(f) \neq f$ or $\theta(g) \neq g$. We say that a generalization is *condensed* if it is not redundant.

A variable in a condensed generalization must occur either at the outermost position or as an argument of a constant. This bounds the depth of terms.

Examples 4.12 and 4.13 illustrate that we must limit the number of times subterms can appear. The solution is to disallow most substitutions which eliminate subterms.

Definition 5.2 A substitution $\theta : t[f(\overline{u})] \to a$ is said to *eliminate* u_k if $\theta(f) = \lambda\overline{x}.M$ and x_k does not occur in M.

That is, a subterm of $f(\overline{u})$ is eliminated if $\theta(f)$ is independent of the corresponding abstraction. This is a generalization of the definition of elimination introduced in [13].

Definition 5.3 A subterm u_k of $t[H(\overline{u})]$ is *uneliminable* if

i. $u_k \notin \mathcal{BV}(t)$, or

ii. $u_k \in \mathcal{BV}(t)$ and $u_k = u_{k'}$ for $k' \neq k$.

Definition 5.4 A generalization $\theta : t \to a$ in $\mathbf{G}(a)$ is *cluttered* if for some $f \in dom(\theta)$, $\theta(f)$ eliminates an uneliminable subterm.

By disallowing cluttered generalizations, we bound the width of terms. However, some generalizations which eliminate bound variables *are* allowed so that $\lambda xy.x$ and $\lambda xy.y$ can be generalized (using $\lambda xy.f(x, y)$).

Definition 5.5 A generalization $g \in \mathbf{G}(a)$ is said to be *relevant* if it is *condensed* and *not cluttered*. Let $\mathbf{R}(a)$ be the full subcategory of $\mathbf{G}(a)$ consisting of relevant generalizations. Similarly, let $\mathbf{R}(a, b)$ be the full subcategory of $\mathbf{G}(a, b)$ consisting of pairs of relevant generalizations.

Examples 5.6 The following generalizations are *irrelevant*:

$$\begin{tikzcd}
\lambda x.\mathsf{last}(\mathsf{rev}(x)) & & \lambda x.\mathsf{head}(x) \\
& \lambda x.f(g(x)) \arrow[ul, "{[\mathsf{last}, \mathsf{rev}]}"] \arrow[ur, "{[\pi, \mathsf{head}]}"'] &
\end{tikzcd}$$

This is redundant because of the bindings of f and g.

$$\begin{tikzcd}
\mathsf{append}(\mathsf{x}, \mathsf{l}) & & \mathsf{cons}(\mathsf{x}, \mathsf{l}) \\
& g(\mathsf{cons}(\mathsf{x}, \mathsf{l})) \arrow[ul, "{[\lambda z.\mathsf{append}(\mathsf{x}, \mathsf{l})]}"] \arrow[ur, "{[\pi]}"'] &
\end{tikzcd}$$

This is cluttered because the subterm $\mathsf{cons}(\mathsf{x}, \mathsf{l})$ is not eliminable.

$$\begin{tikzcd}
\lambda x.\mathsf{zerop}(x) & & \lambda x.x = x \\
& \lambda x.f(x, x) \arrow[ul, "{[\mathsf{zerop} \circ \pi_1^2]}"] \arrow[ur, "{[=]}"'] &
\end{tikzcd}$$

This is cluttered because x is not eliminable.

If we ignore renamings,

Lemma 5.7 $\mathbf{R}(a)$ is finite.

This is because the number of constants is limited by the size of a, which limits the number of free variables (since each must be separated by a constant), and the sum of the two limits the number of bound variables. Since $\mathbf{R}(a, b)$ contains only pairs of objects from both $\mathbf{R}(a)$ and $\mathbf{R}(b)$,

Corollary 5.8 $\mathbf{R}(a, b)$ is finite.

Since the most specific generalization may not be unique, we define the set of maximally specific generalizations:

Definition 5.9 $\mathrm{MSG}(a, b)$ is the least set of generalizations in $\mathbf{R}(a, b)$ such that $\forall g \in \mathbf{R}(a, b), \exists g' \in \mathrm{MSG}(a, b)$ such that $g \to g'$ (up to an isomorphism).

Note that the least set exists because if $g \to g_1'$ and $g \to g_2'$ where $g_1' \leftrightarrow g_2'$, then g_1' and g_2' are isomorphic by Lemma 4.7.

Examples 5.10 Some maximal (relevant) generalizations:

$$\begin{tikzcd}
\lambda x.\mathsf{len}(\mathsf{nodes}(x)) & & \lambda x.\mathsf{len}(\mathsf{fringe}(x)) \\
& \lambda x.\mathsf{len}(f(x)) \arrow[ul, "{[\mathsf{nodes}]}"'] \arrow[ur, "{[\mathsf{fringe}]}"] &
\end{tikzcd}$$

This is maximally specific because len is the only common constant.

$$\begin{tikzcd}
\mathsf{sqr}(2) & & \mathsf{sqr}(\mathsf{sqr}(2)) \\
& f(\mathsf{sqr}(2)) \text{ or } \mathsf{sqr}(g(2)) \arrow[ul, "{[\pi]}"'] \arrow[ur, "{[\mathsf{sqr}]}"] &
\end{tikzcd}$$

This illustrates how multiple maximal objects can arise when there are different possible pairings.

$$\begin{tikzcd}
\lambda x.x + x + \mathsf{C}(2) & & \lambda x.x - \mathsf{C}(2) * x - \mathsf{C}(2) \\
& \lambda x.f(x, x, \mathsf{C}(2)) \arrow[ul, "{[\lambda abc.a + b + c]}"'] \arrow[ur, "{[\lambda abc.a{-}c{*}b{-}\mathsf{C}(2)] \text{ or } [\lambda abc.a{-}c{*}b{-}c]}"] &
\end{tikzcd}$$

Different ways of instantiating can also lead to multiple generalizations. Note that there are more generalizations as well.

5.1 Properties of R

G is not a preorder because its morphisms are not always unique. In this section, we show that **R** is a preorder. This property is interesting in itself and also helps in showing the correctness of our algorithm to compute the complete set of most specific generalizations. All the results of this section extend to the binary case (and multiple term cases) because the morphisms of $\mathbf{R}(a, b)$ are a subset of the morphisms of $\mathbf{R}(a)$ and $\mathbf{R}(b)$.

Lemma 5.11 Whenever g_1, $g_2 \in \mathbf{R}(a)$ and $\rho : g_1 \to g_2$, $\rho : src(g_1) \to src(g_2)$ is relevant.

Proof Let g_1 be $\theta_1 : t_1 \to a$ and g_2 be $\theta_2 : t_2 \to a$. If $\rho : t_1 \to t_2$ is not relevant, ρ eliminates a subterm w of t_1. But then $\theta_2 \circ \rho$ eliminates w; this contradicts $\theta_2 \circ \rho = \theta_1$. $\boldsymbol{\Lambda}$

Theorem 5.12 $\mathbf{R}(a)$ is a preorder.

We need to show that there is at most one morphism between any two generalizations.[4] Consider the commutative diagram

$$\begin{tikzcd}
u \arrow[r, "\sigma"] & a \\
t[f(\overline{w})] \arrow[u, shift right, "\rho_1"] \arrow[u, shift left, "\rho_2"'] \arrow[ur, "\theta"'] &
\end{tikzcd}$$

in $\mathbf{R}(a)$ and let $f(\overline{w})$ be the outermost subterm of t such that f is a free variable and $\rho_1(f) \neq \rho_2(f)$. Observe that since $\rho_1 : t \to u$ and $\rho_2 : t \to u$ are relevant, there is at least one occurrence of $\rho_1(f(\overline{w}))$ and $\rho_2(f(\overline{w}))$ in u. Also observe that this occurrence must be the same for both ρ_1 and ρ_2 since f is the outermost variable for which ρ_1 and ρ_2 differ. Call this occurrence u'. The key lemma for showing that f does not exist is

Lemma 5.13 If $\rho_1(f) = \pi_i^n$, then $\rho_2(f) = \pi_i^n$.

Proof Since $\rho_1 : t \to u$ is relevant, each $\overline{w}$ other than w_i must be eliminable, so they are all projections different from each other and different from w_i. There are three cases:

1. *head*(w_i) is a constant: w_i is uneliminable, so $\rho_2(f) = \pi_i^n$.
2. *head*(w_i) is a free variable: this case is impossible because $\rho_1 : t \to a$ is condensed.
3. $w_i \in \mathcal{BV}(t)$: $u' = w_i$, so since there is no other $w_j = w_i$, $\rho_2(f) = \pi_i^n$.

$\mathbf{\Lambda}$

This along with the existence of a $u' = \rho_1(f(\overline{w}))$ gives us

Lemma 5.14 *head*$(\rho_1(f)) =$ *head*$(\rho_2(f))$.

Proof of 5.12 We show $\rho_1(f) = \rho_2(f)$. Suppose $\rho_1(f) = \lambda\overline{x}.K(\overline{M})$ for some constant K. Then $\rho_2(f)$ is $\lambda\overline{x}.K(\overline{N})$ by Lemma 5.14, and we use induction on the depth of substitutions (using a multiset ordering) to show $\overline{M} = \overline{N}$ by constructing the commutative diagram

$$\begin{tikzcd}
u \arrow[r, "\sigma"] & a \\
v \arrow[u, shift right, "\rho_1'"] \arrow[u, shift left, "\rho_2'"'] \arrow[ur, "\theta'"'] & \\
t[f(\overline{w})] \arrow[u, "\phi"] \arrow[uur, "\theta"'] &
\end{tikzcd}$$

[4] Note that renamings are *not* allowed.

in $\mathbf{R}(a)$ such that $\rho_1 = \rho'_1 \circ \phi$ and $\rho_2 = \rho'_2 \circ \phi$. The following function is used to ensure $\theta' : v \to a$ is relevant:

Definition 5.15

$$\begin{array}{l} \mathit{projected}(\overline{x_n}, t) = \\ \qquad \text{the } \langle x_{j_1}, \ldots, x_{j'_n} \rangle \text{ such that each } x_{j_k} \in \{\overline{x_n}\}, \text{ all } j_k < j_{k+1}, \text{ and } x_{j_k} \text{ occurs in } t \end{array}$$

Then let

$$\begin{array}{rcl} \theta(f) & = & \lambda\overline{x_n}.K(\overline{P}) \\ \nu_i & = & \mathit{projected}(\overline{x_n}, M_i) \cup \mathit{projected}(\overline{x_n}, N_i) \\ \phi & = & \{f \mapsto \lambda\overline{x_n}.K(h_1(\nu_1), \ldots, h_m(\nu_m))\} \\ \rho'_1 & = & \rho_1 \setminus f \cup \{h_1 \mapsto \lambda\nu_1.M_1, \ldots, h_m \mapsto \lambda\nu_m.M_m\} \\ \rho'_2 & = & \rho_2 \setminus f \cup \{h_1 \mapsto \lambda\nu_1.N_1, \ldots, h_m \mapsto \lambda\nu_m.N_m\} \\ \theta' & = & \theta \setminus f \cup \{h_1 \mapsto \lambda\nu_1.P_1, \ldots, h_m \mapsto \lambda\nu_m.P_m\} \end{array}$$

(where each $\overline{h_m}$ is a free variable occurring nowhere else). Note that $\mathit{projected}(\overline{x_n}, P_i)$ must be a subsequence of ν_i since σ cannot introduce abstractions. Furthermore, if N_i eliminates x_j in ν_i, then w_j is eliminable. This is because if x_j does not appear in N_i, it must be eliminated by σ from M_i, so it must be in the scope of a free variable f'' in M_i. Since $\sigma : u \to a$ is uncluttered and w_j is eliminated by $\sigma(f'')$, w_j must be eliminable. Thus $\theta' : v \to a$ is relevant and we can use induction to show $\rho'_1(f) = \rho'_2(f)$.

A similar argument is used when the head of both $\rho_1(f)$ and $\rho_2(f)$ is a free variable, g, except that the details must be modified to ensure $\theta' : v \to a$ is condensed. Observe that the arguments to g must be rigid terms (unless $\rho_1(f) = f$ and $\rho_1(g) = g$, in which case $\rho_2(f) = f$ and $\rho_2(g) = g$ because $\rho_2 : v \to u$ is relevant). Thus $\rho_1(f) = \lambda\overline{x_n}.g(\overline{M_m})$ and $\rho_2(f) = \lambda\overline{x_n}.g(\overline{N_m})$ where $M_i = G_i(\overline{r_{i,p_i}})$ and $N_i = H_i(\overline{s_{i,q_i}})$. We first show that for each k, $H_k = G_k$. Since $\sigma \circ \rho_1 = \theta = \sigma \circ \rho_2$, $\sigma(g \circ \langle \ldots, G_k, \ldots \rangle) = \sigma(g \circ \langle \ldots, H_k, \ldots \rangle)$ and so $\sigma(G_k) = \sigma(H_k)$. Thus $G_k = H_k$ since both are rigid.

We use induction to show that the rest of $\rho_1(f)$ and $\rho_2(f)$ are the same. Assuming $\theta(f)$ is $\lambda\overline{x_n}.g(H_1(r'_{1,1}, \ldots, r'_{1,p_1}), \ldots, H_m(r'_{m,1}, \ldots, r'_{m,p_m}))$, let

$$\begin{array}{rcl} \nu_{i,j} & = & \mathit{projected}(\overline{x_n}, r_{i,j}) \cup \mathit{projected}(\overline{x_n}, s_{i,j}) \\ \phi & = & \{f \mapsto \lambda\overline{x_n}.g(H_1(h_{1,1}(\nu_{1,1}), \ldots, h_{1,p_1}(\nu_{1,p_1})), \ldots, \\ & & \qquad H_m(h_{m,1}(\nu_{m,1}), \ldots, h_{m,p_m}(\nu_{m,p_m})))\} \\ \rho'_1 & = & \rho_1 \setminus f \cup \{h_{1,1} \mapsto \lambda\nu_{1,1}.r_{1,1}, \ldots, h_{m,p_m} \mapsto \lambda\nu_{m,p_m}.r_{m,p_m}\} \\ \rho'_2 & = & \rho_2 \setminus f \cup \{h_{1,1} \mapsto \lambda\nu_{1,1}.s_{1,1}, \ldots, h_{m,p_m} \mapsto \lambda\nu_{m,p_m}.s_{m,p_m}\} \\ v & = & \phi(t) \\ \theta' & = & \theta \setminus f \cup \{h_{1,1} \mapsto \lambda\nu_{1,1}.r'_{1,p_1}, \ldots, h_{m,p_m} \mapsto \lambda\nu_{m,p_m}.r'_{m,p_m}\} \end{array}$$

Again $\rho'_1 = \rho'_2$ by induction, hence $\rho_1 = \rho_2$.

Since morphisms are unique and θ_{id} is a morphism from any generalization to itself,

Corollary 5.16 $\mathbf{R}(a)$ is a partial order.

This allows us to introduce the following notation:

Definition 5.17 Whenever $g_1 \to g_2$ is in $\mathbf{R}(a)$, we say g_1 is *less specific* (or, equivalently, *more general*) than g_2. This is written as $g_1 \leq g_2$. Furthermore, we write $g_1 < g_2$ if $g_2 \to g_1$ is not in $\mathbf{R}(a)$.

6 Computing MSG

$\mathbf{R}(a, b)$ is finite, so since second-order matching is decidable and λ-terms are recursively enumerable, we can compute $\mathrm{MSG}(a, b)$ by generating $\mathbf{R}(a, b)$ and comparing all its objects against each other. Thus, $\mathrm{MSG}(a, b)$ is computable, albeit inefficiently. A more practical algorithm is suggested by the observation that when the substitutions contain a common subterm, then they can be made more specific by factoring out the common term.

The steps for specializing generalizations of a and b are given by the following rewrite relation $\longrightarrow$. The algorithm is restricted to generalizing ground terms; non-ground terms can be handled by "freezing" the variables; that is, replacing them by unique constants. The $\longrightarrow$ steps maintain the invariants

$$\begin{array}{l} \theta_1 : t \to a \\ \theta_2 : t \to b \\ \langle \theta_1 : t \to a,\ \theta_2 : t \to b \rangle \text{ is relevant} \\ \text{if } g_1 \longrightarrow g_2,\ g_1 < g_2 \end{array}$$

To compute $\mathrm{MSG}(a, b)$, we start with the initial object of $\mathbf{R}(a, b)$ and continue specializing the generalization until no $\longrightarrow$ step is applicable. To simplify the notation, we represent each generalization $\langle \theta_1 : t \to a,\ \theta_2 : t \to b \rangle$ by the triple $\langle t, \theta_1, \theta_2 \rangle$.

Delete-variable Variables with the same binding in both substitutions can be removed:

$$\langle t, \theta_1 \cup \{f \mapsto M\}, \theta_2 \cup \{f \mapsto M\} \rangle \quad \longrightarrow \quad \langle \{f \mapsto M\}(t), \theta_1, \theta_2 \rangle$$

Merge Likewise, variables with the same bindings within each substitution can be merged:

$$\begin{array}{lcl} \langle t, & & \langle \{g \mapsto f\}(t), \\ \quad \theta_1 \cup \{f \mapsto M, g \mapsto M\}, & \longrightarrow & \quad \theta_1 \cup \{f \mapsto M\}, \\ \quad \theta_2 \cup \{f \mapsto N, g \mapsto N\} \rangle & & \quad \theta_2 \cup \{f \mapsto N\} \rangle \end{array}$$

Delete-abstraction Subterms which are not projected by either substitution can be eliminated:

$$\begin{array}{lcl} \langle t, & & \langle \{f \mapsto \lambda \overline{x} z \overline{y}. f'(\overline{x}, \overline{y})\}(t), \\ \quad \theta_1 \cup \{f \mapsto \lambda \overline{x} z \overline{y}. M\}, & \longrightarrow & \quad \theta_1 \cup \{f' \mapsto \lambda \overline{x}\overline{y}. M\}, \\ \quad \theta_2 \cup \{f \mapsto \lambda \overline{x} z \overline{y}. N\} \rangle & & \quad \theta_2 \cup \{f' \mapsto \lambda \overline{x}\overline{y}. N\} \rangle \end{array}$$

where z does not occur in either M or N.

Factor-constant Constants that appear in both substitutions can be factored. This step is complicated because it must introduce new function variables for generalizing the subterms and it must not create cluttered terms.

$$\begin{array}{lcl}
 & & \langle\{f \mapsto \lambda\overline{x}.f'(K(h_1(\nu_1),\ldots,h_n(\nu_n)),\nu_0)\}(t), \\
\langle t, & & \quad \theta_1 \cup \{f' \mapsto \lambda z\nu_0.M[\forall\alpha \in \hat{\alpha}, \alpha \leftarrow z], \\
\quad \theta_1 \cup \{f \mapsto \lambda\overline{x}.M[K(\overline{u})]\}, & \longrightarrow & \qquad\quad h_1 \mapsto \lambda\nu_1.u_1,\ \ldots,\ h_n \mapsto \lambda\nu_n.u_n\}, \\
\quad \theta_2 \cup \{f \mapsto \lambda\overline{x}.N[K(\overline{v})]\}\rangle & & \quad \theta_2 \cup \{f' \mapsto \lambda z\nu_0.N[\forall\beta \in \hat{\beta}, \beta \leftarrow z], \\
 & & \qquad\quad h_1 \mapsto \lambda\nu_1.v_1,\ \ldots,\ h_n \mapsto \lambda\nu_n.v_n\}\rangle
\end{array}$$

where

K is a constant,
n is the arity of K,
h_i occurs no where else (for $1 \leq i \leq n$),
$\hat{\alpha}$ is a nonempty subset of the positions at which $K(\overline{u})$ occurs in M,
$\hat{\beta}$ is a nonempty subset of the positions at which $K(\overline{v})$ occurs in N,
ν_0 $= \mathit{projected}(\overline{x}, M[\forall\alpha \in \hat{\alpha}, \alpha \leftarrow z]) \cup \mathit{projected}(\overline{x}, N[\forall\beta \in \hat{\beta}, \beta \leftarrow z])$
(see Definition 5.15 for *projected*), and
ν_k $= \mathit{projected}(\overline{x}, u_k) \cup \mathit{projected}(\overline{x}, v_k)$.

If the new θ_1 or θ_2 of some h_k (or f') would eliminate an uneliminable subterm, then this step is not applicable (with the chosen $\hat{\alpha}$ and $\hat{\beta}$) because it would form a cluttered generalization.

Factor-abstraction Repeated bound variables can be factored in much the same way as constants except that there is no need to introduce new free variables:

$$\begin{array}{lcl}
\langle t, & & \langle\{f \mapsto \lambda\overline{x}.f'(x_i,\overline{x})\}(t), \\
\quad \theta_1 \cup \{f \mapsto \lambda\overline{x}.M[x_i]\}, & \longrightarrow & \quad \theta_1 \cup \{f' \mapsto \lambda z\overline{x}.M[\forall\alpha \in \hat{\alpha}, \alpha \leftarrow z]\}, \\
\quad \theta_2 \cup \{f \mapsto \lambda\overline{x}.N[x_i]\}\rangle & & \quad \theta_2 \cup \{f' \mapsto \lambda z\overline{x}.N[\forall\beta \in \hat{\beta}, \beta \leftarrow z]\}\rangle
\end{array}$$

where x_i is in $\overline{x}$, x_i occurs in at least one other position in *both* M and N, and $\hat{\alpha}$ and $\hat{\beta}$ are proper, nonempty subsets of the positions at which x_i occurs. ($\hat{\alpha}$ and $\hat{\beta}$ must be proper subsets so that the new generalization is not cluttered.)

Using $\longrightarrow$, the set MSG can be computed by *gen* defined as

$$\mathit{gen}(a,b) \quad = \quad \{g \mid \langle f,[a],[b]\rangle \longrightarrow^* g, \text{ and } \nexists g'.g \longrightarrow g'\}$$

where $\longrightarrow^*$ is the transitive closure of $\longrightarrow$.

This algorithm is expensive because it requires exponential time and recomputes the same generalizations in different ways. Furthermore, some pairs of terms have an exponential number of generalizations, so there is no polynomial-time algorithm based on the size of the input. It is not yet clear if a polynomial-time algorithm exists based on the number of generalizations.

The proof this algorithm's correctness depends upon showing that the set of $\longrightarrow$ rules completely specifies when one generalization is strictly less instantiated than another. First we give some lemmas:

Lemma 6.1 If $g_1 \in \mathbf{R}(a,b)$ and $g_1 \longrightarrow g_2$, then $g_2 \in \mathbf{R}(a,b)$.

Lemma 6.2 Whenever g_1, $g_2 \in \mathbf{R}(a,b)$ and $g_1 \longrightarrow g_2$, $g_1 < g_2$.

Proof Observe that each step is of the form $\langle t, \theta_1, \theta_2\rangle \longrightarrow \langle \rho(t), \theta_1', \theta_2'\rangle$ where ρ is a generalization morphism. This shows $g_1 \leq g_2$. Furthermore, ρ is not a renaming substitution, so by Theorem 5.12 there is no $\rho_{op} : g_2 \to g_1$. $\mathbf{\Lambda}$

Finally, we show that $\longrightarrow$ steps do not reduce the number of possible generalizations. That is, given a specific generalization, the set of $\longrightarrow$ steps completely covers all maximal generalizations which are more instantiated than the given one.

Lemma 6.3 Whenever $g_v' \in \mathrm{MSG}(a,b)$, $g_t \in \mathbf{R}(a,b)$, and $g_t < g_v'$, there is a $g_u \in \mathbf{R}(a,b)$ and a $g_v \cong g_v'$ such that $g_t \longrightarrow g_u$ and $g_u \leq g_v$.

Proof Assume

$$\begin{aligned} g_t &= \langle \theta_1 : t \to a,\ \theta_2 : t \to b\rangle \\ g_u &= \langle \sigma_1 : u \to a,\ \sigma_2 : u \to b\rangle \\ g_v &= \langle \phi_1 : v \to a,\ \phi_2 : v \to b\rangle \end{aligned}$$

Then the following diagram illustrates this lemma:

$$\begin{tikzcd}
a & v \arrow[l, "\phi_1"'] \arrow[r, "\phi_2"] & b \\
& u \arrow[ul, "\sigma_1"] \arrow[u, "\rho_u"] \arrow[ur, "\sigma_2"'] & \\
& t \arrow[u, "\rho_t"] \arrow[uul, "\theta_1"] \arrow[uur, "\theta_2"'] \arrow[uu, bend right, "\rho_v"'] &
\end{tikzcd}$$

Choose an $f \in dom\rho_v$ such that $\rho_v(f)$ is not a renaming substitution unless all substitutions are renamings. Let

$$\begin{aligned} \rho_v(f) &= \lambda\overline{x}.H(\overline{w}) \\ \theta_1(f) &= \lambda\overline{y}.M \\ \theta_2(f) &= \lambda\overline{y}.N \end{aligned}$$

Furthermore, assume that if $\lambda\overline{x}.H(\overline{w})$ is a renaming substitution, then $H \in dom\rho_v$ and $\rho_v(H) = H$. Observe that such an f exists because $g_t < g_v$. We will show that for any $\lambda\overline{x}.H(\overline{w})$, there is a $\longrightarrow$-step which generates an appropriate g_u. In most cases, we only identify which step is applicable;

refer to the algorithm for the details of constructing g_u and ρ_u. Note that if there is a step to create g_u, then $g_u \in \mathbf{R}(a, b)$ by Lemma 6.1.

There are three cases based on the form of H.

1. H is x_i in $\overline{x}$: $\theta_1(f) = \phi_1(\rho_v(f)) = \pi_i = \phi_2(\rho_v(f)) = \theta_2(f)$, so **Delete-variable** is applicable.

2. H is a constant K: $\mathit{head}(\theta_1) = \mathit{head}(\phi_1(\rho_v(f))) = K = \mathit{head}(\phi_2(\rho_v(f))) = \mathit{head}(\theta_2)$, so **Factor-constant** is applicable.

3. H is a free variable (say g): Since we are only interested in finding an isomorphism of g'_v, we can reorder the arguments to g as $g(x_1, \ldots, x_k, w_{k+1}, \ldots, w_n)$ (with corresponding reorderings to $\phi_1(g)$ and $\phi_2(g)$) such that k is the smallest integer for which $w_{k+1} \neq x_{k+1}$.

   If $k = n$, then $\theta_1(f) = \phi_1(\rho_v(f)) = \phi_1(g)$ and by assumption $\theta_1(g) = \phi_1(\rho_v(g)) = \phi_1(g)$. Thus $\theta_1(f) = \theta_1(g)$. Likewise, $\theta_2(f) = \theta_2(g)$. Hence the **Merge** step is applicable.

   If $k < n$, then there are four cases depending upon the form of w_{k+1}:

   (a) w_{k+1} is flexible: this would make g_v redundant, a contradiction.

   (b) $w_{k+1} = x_i$ for $i \leq k$: y_i occurs more than once in both M and N and so a **Factor-abstraction** step is applicable. Pick $\hat{\alpha}$ and $\hat{\beta}$ such that the occurrences of y_i in $\sigma_1(f')$ and $\sigma_2(f')$ match those in $\phi_1(g)$ and $\phi_2(g)$.

   (c) $w_{k+1} = x_i$ for $i > k + 1$: y_{k+1} does not occur in either M or N and so a **Delete-abstraction** step is applicable.

   (d) $w_{k+1} = K(\overline{s})$ where K is a constant: because g_v is not cluttered, K must occur in both M and N, thus a **Factor-constant** step is applicable. Again, pick $\hat{\alpha}$ and $\hat{\beta}$ such that the occurrences of K in $\sigma_1(f')$ and $\sigma_2(f')$ match those in $\phi_1(g)$ and $\phi_2(g)$.

$\mathbf{\Lambda}$

Theorem 6.4 (Soundness) If $g \in \mathit{gen}(a, b)$, then $g \in \mathrm{MSG}(a, b)$.

Proof By Lemmas 6.1 and 6.2, if $g \in \mathit{gen}(a, b)$ then $g \in \mathbf{R}(a, b)$. g is maximal by Lemma 6.3. $\mathbf{\Lambda}$

Theorem 6.5 (Completeness) If g is in $\mathrm{MSG}(a, b)$, then there is a generalization g' in $\mathit{gen}(a, b)$ which is isomorphic to g.

Proof By Observation 4.8, we know that g_0 is less specific than any generalization of a and b, so by Lemma 6.3 we know that for any g there is a sequence of $\longrightarrow$ steps from g_0 to some g' isomorphic to g. This sequence is finite because $<$ is well-founded and $g_1 \longrightarrow g_2$ implies $g_1 < g_2$. $\mathbf{\Lambda}$

7 Conclusion

We provide a framework for unsolved problem of generalizing second-order terms. Our solution is based on viewing the structure of terms as a category rather than a partial order. The categorical view allows us to capture *how* one term generalizes another, which is not possible in the conventional structure of complete lattices [14, 16].

Second-order generalization seems eminently useful for generalizing first-order terms in a useful fashion. For instance, A and $\mathsf{A} \wedge \mathsf{B}$ have the maximal generalization

$$\langle [\pi] : f(\mathsf{A}) \rightarrow \mathsf{A},\ [\lambda x.x \wedge \mathsf{B}] : f(\mathsf{A}) \rightarrow \mathsf{A} \wedge \mathsf{B} \rangle$$

showing that A is replaced by a conjunction in going to $\mathsf{A} \wedge \mathsf{B}$. This information is lost in the corresponding first-order most specific generalization. Similarly, going to third and higher orders would improve the quality of generalization. More importantly, base terms of higher orders also necessitate going to higher orders. We intend to pursue this in future work.

Acknowledgements

The authors would like to thank John Gray for useful discussions on category theory and the anonymous reviewers for a number of helpful comments. We would also like to thank Paul Taylor for the macros used to generate the diagrams.

References

[1] Michael A. Arbib and Ernest G. Manes. *Arrows, Structures, and Functors: The Categorical Imperative.* Associated Press, New York, 1975.

[2] R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

[3] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–58, 1940.

[4] R. L. Constable, *et. al. Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, New York, 1986.

[5] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, Cambridge, 1989.

[6] Joseph A. Goguen. What is unification? A categorical view of substitution, equation, and solution. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, chapter 6, pages 217–261. Academic Press, 1989.

[7] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings, Symposium on Logic in Computer Science*, pages 194–204, Ithaca, New York, June 1987. IEEE Computer Society Press.

[8] Gérard Huet. Résolution d'équations dans des langages d'order $1, 2, \ldots, \omega$ (these d'etat), December 1976.

[9] F. William Lawvere. Functional semantics of algebraic theories. In *National Academy of Sciences*, 1963.

[10] Lawrence C. Paulson. Natural deduction as higher-order resolution. *Journal of Logic Programming*, 3:237–258, 1986.

[11] Frank Pfenning. Elf: A language for logic definition and verified meta-programming. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 313–322, Pacific Grove, California, June 1989. IEEE Computer Society Press.

[12] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[13] T. Pietrzykowski. A complete mechanization of second-order type theory. *Journal of the ACM*, 20(2):333–365, April 1973.

[14] Gordon D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, chapter 8, pages 153–163. American Elsevier, New York, 1970.

[15] U. S. Reddy. Transformational derivation of programs using the Focus system. *SIGSOFT Software Engineering Notes*, 13(5):163–172, November 1988. Proceedings, ACM SIGSOFT/SIGPLAN Third Software Engineering Symposium on Practical Software Development Environments; also published as *SIGPLAN Notices*, Feb. 1989.

[16] John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In *Machine Intelligence*, volume 5, chapter 7, pages 135–151. Edinburgh Univ. Press, Edinburgh, 1970.