CSci 8980, Fall 2012
Specifying and Reasoning About Computational Systems
The Unification of Lambda Terms

Gopalan Nadathur

Department of Computer Science and Engineering
University of Minnesota

Lectures in Fall 2012

The Need for Unification

Recall the $\forall L$ and $\exists R$ rules in the reduced proof system

$$\frac{\Sigma; \mathcal{P} \vdash B[t/x] \qquad t \text{ is a } \Sigma\text{-term}}{\Sigma; \mathcal{P} \vdash \exists x\ B}\ \exists R$$

$$\frac{\Sigma; \mathcal{P} \xrightarrow{D[t/x]} A \qquad t \text{ is a } \Sigma\text{-term}}{\Sigma; \mathcal{P} \xrightarrow{\forall x\ D} A}\ \forall L$$

These rules require us to guess a suitable term before we have any information of how to do this

These rules are therefore not quite suitable for proof search

The standard technique is to overcome this obstacle is to delay such choices using *logic variables* and unification

Logic Variables and Unification

Logic variables, denoted by tokens starting with uppercase letters, are place-holders for actual terms to be decided later

Using them, the $\forall L$ and $\exists R$ rules become

$$\frac{\Sigma; \mathcal{P} \vdash B[X/x] \qquad X \text{ is a } \Sigma\text{-term}}{\Sigma; \mathcal{P} \vdash \exists x\ B}\ \exists R$$

$$\frac{\Sigma; \mathcal{P} \xrightarrow{D[X/x]} A \qquad X \text{ is a } \Sigma\text{-term}}{\Sigma; \mathcal{P} \xrightarrow{\forall x\ D} A}\ \forall L$$

Actual substitutions will be dictated by the needs of the *init* rule

$$\frac{}{\Sigma; \mathcal{P} \xrightarrow{A'} A}\ \textit{init}, A \text{ and } A' \text{ unify}$$

Some important points to note

- substitutions generated by *init* must be applied to the entire derivation
- Instantiations must respect signature constraints

Unification Under a Quantifier Prefix

A nice conceptual treatment of signature constraints results from moving quantifiers to the derivation level

These quantifiers then become a prefix to unification problems

Specifically, a unification problem has the structure

$$\mathcal{Q}_1 x_1\ \ldots\ \mathcal{Q}_n x_n\ [t_1 = s_1 \wedge \ldots \wedge t_m = s_m]$$

where $\mathcal{Q}_i$ is $\forall$ or $\exists$

Such a problem is solved by a substitutions for the existentially quantified variables that respects quantifier scopes

For uniformity, top-level constants are represented in this framework by outermost universal quantifiers

Unification Under a Quantifier Prefix (Continued)

The prefixed form represents the unification problem along only *one branch* of a proof tree

Note that prenexing is not a sound principle:

$$\forall y\ \exists x\ [y = y \wedge x = x] \quad \text{and} \quad (\forall y\ [y = y]) \wedge (\exists x\ [x = x])$$

do not have the same solutions

it is therefore unclear how to transform a problem with the structure

$$(\forall x\ \exists z\ ...) \wedge (\forall y\ \exists w\ ...)$$

into a prefixed form that preserves the solutions

Thus, we will in general be concerned with solving a *conjunction* of unification problems with shared prefixes

Unifiers versus Solutions to Unification Problems

To be logically correct, solutions must be *closed*, i.e., they must only contain suitable universally quantified variables

However, often it is better to stop short of exhibiting actual solutions

For example, consider

$$\forall a\ \forall b\ \exists x\ \exists y\ x = y$$

Here x and y must be identical *and* must be one of a or b

However, making the last choice now is premature

Substitutions that solve all the equations but do not necessarily produce actual terms are called *unifiers*

We will focus on finding unifiers, noting that eventually exhibiting an inhabitant is important for the logic

Simplifying the Quantifier Prefix via Raising

Often we will treat unification problems in which the prefix has a $\forall\exists\forall$ or even a $\exists\forall$ form

Any arbitrary unification problem can be put in this form by a technique called *raising*

Specifically, raising moves existential quantifiers using the following kind of transformation

$$\forall x\ \exists y\ B \quad \rightsquigarrow \quad \exists f\ \forall x\ B[(f\ x)/y]$$

The end point for such a transformation is a $\exists\forall$ prefix

However, it is often convenient to leave the outermost universal quantifiers representing top-level constants implicit

The Correctness of Raising

Raising is justified purely on proof-theoretic principles

Proposition

There is a one-to-one correspondence between the solutions to $\forall x\ \exists y\ B$ and $\exists f\ \forall x\ B[(f\ x)/y]$

Proof

We show how to go to and from solutions to the two problems

- Suppose that $\{y \mapsto t\}$ solves the first problem
  Consider the substitution $\lambda x\ t$ for f
- Suppose $\{f \mapsto \lambda z\ t\}$ solves the second problem
  Consider the substitution of $t[x/z]$ for y

Note also that a back-and-forth transformation preserves the original substitution

Note: Raising preserves the property of being in L_λ

Comparison with Skolemization

Dual to raising that moves existential quantifiers inwards using

$$\exists x\ \forall y\ P \rightsquigarrow \forall f\ \exists x\ P[(f\ x)/y]$$

Intuitively, the dependency of $(f\ x)$ on x prevents it from appearing in a substitution for x

Used in settings where there are no higher-order variables

However, there are problems with Skolemization

- Assume $z : (a{\rightarrow}b){\rightarrow}a$, $x : a$, $y : b$, $f : a{\rightarrow}b$ and contrast

  $$\forall z\ \exists x\ \forall y\ \top \quad \text{with} \quad \forall z\ \forall f\ \exists x\ \top$$

  The second has a solution whereas the first does not
- Assume $z : a, g : b{\rightarrow}a, x : a, y : b, f : a{\rightarrow}b$ and contrast

  $$\forall z\ \forall g\ \exists x\ \forall y\ \top \quad \text{with} \quad \forall z\ \forall g\ \forall f\ \exists x\ \top$$

  The first has one solution whereas the second has an infinite number, i.e., it is not clear how to relate solutions

A Simplified Form for Equations

Since we solve equations modulo λ-conversion, we can assume equations have the form

$$\lambda x_1\ \ldots \lambda x_n\ (h_1\ t_1\ \ldots\ t_n) = \lambda x_1\ \ldots \lambda x_n\ (h_2\ s_1\ \ldots\ s_k)$$

where the body has atomic type and h_1 and h_2 are existential, universal or lambda-bound variables

Moreover, using the fact that the solutions for

$$\overline{\mathcal{Q}}\,[\lambda x\ t = \lambda x\ s] \quad \text{and} \quad \overline{\mathcal{Q}}\,\forall x\ [t = s]$$

are identical, we can assume that the binders are empty

We will base our analysis only on problems in this form

Also, we will use the terminology of *rigid-rigid*, *rigid-flexible* and *flexible-flexible* pairs

Properties of Higher-Order Unification

The general problem possesses many properties that are at least daunting from a computational perspective

- We cannot guarantee that there is *one* unifier that covers all others, e.g. consider

  $$\forall a_i\ \forall g_{\ i\rightarrow i\rightarrow i}\exists F_{i\rightarrow i}\ [(F\ a) = (g\ a\ a)]$$

  Four incomparable unifiers exist for this problem
- "Complete sets of unifiers" may be infinite, e.g. consider

  $$\forall u_{i\rightarrow i}\ \exists F_{i\rightarrow i}\ \forall w_i\ [u\ (F\ w) = F\ (u\ w)]$$

  Any substitution of the form $\{F \mapsto \lambda y(u\ \ldots\ (u\ y)\ldots)\}$ is a unifier
- Unifiers cannot always be generated in a non-redundant way

Properties of Higher-Order Unification (Continued)

- Determining whether or not there are unifiers is an undecidable problem

  Finding integer solutions to Diophantine equations can be encoded as a higher-order unification problem

Despite all these "bad" properties, higher-order unification has been found useful in systems like Isabelle, Twelf and λProlog

There are several observations that help resolve the paradox

- A procedure can be described to at least determine unifiability
- The problems that arise in practical settings are well behaved
- In fact many applications can be limited to a specially well behaved class of problems

Simplifying Rigid-Rigid Pairs

A rigid-rigid equation has the form

$$(c_1\ t_1\ \ldots\ t_n) = (c_2\ s_1\ \ldots\ s_m)$$

where c_1 and c_2 are universally quantified variables

Note that substitutions cannot change the heads of either term

Thus, it is not difficult to see

- the problem can have a unifier only if $c_1 = c_2$ and $m = n$
- its unifiers in this case are identical to those for

$$t_1 = s_1 \wedge \ldots \wedge t_n = s_n$$

- Applying this transformation replaces an equation by a finite number of new ones in which the terms are smaller

In short, we can "eliminate" rigid-rigid pairs via a terminating process akin to term reduction for first-order terms

Processing Flexible-Rigid Pairs

Assuming symmetry for the equality symbol, these pairs have the form

$$(F\ t_1\ \ldots\ t_n) = (c\ s_1\ \ldots\ s_m)$$

where F is existentially quantified and c is universally quantified

Clearly a unifier must substitute for F in such a way that the head of the left term becomes c

There are really only two possible ways that this can happen

- the substitution for F directly introduces c in the head
- the substitution projects onto one of the arguments of F in the term that then does the rest

We think of generating this substitution in two kinds of incremental steps called *imitation* and *projection*

Note: One of these steps can be shown to get us closer if a unifier exists but there can also be a loop if there is no unifier

The Imitation Substitution

The pair that we are trying to "solve" has the form

$$(F\ t_1\ \ldots\ t_n) = (c\ s_1\ \ldots\ s_m)$$

where the type of F is $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$ for β an atomic type

Note that an imitation substitution for F can exist only if c is quantified before F

To be truly incremental, such a substitution must only fix the head of the F, leaving all other decisions to later

Let the type of c be $\gamma_1 \rightarrow \cdots \rightarrow \gamma_m \rightarrow \beta$

Let H_i be a fresh variable of type $\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \gamma_i$ for $i \leq m$

The imitation substitution for F, if it exists, then has the form

$$\lambda x_1\ \ldots\ \lambda x_n\ (c\ (H_1\ x_1\ \ldots\ x_n)\ \ldots\ (H_m\ x_1\ \ldots\ x_n))$$

with the proviso that we introduce existential quantifiers for H_i adjacent to the one for F

The Projection Substitutions

Again, the pair that we are trying to "solve" has the form

$$(F\ t_1\ \ldots\ t_n) = (c\ s_1\ \ldots\ s_m)$$

where the type of F is $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$ for β an atomic type

Let α_i be $\gamma_1 \rightarrow \cdots \rightarrow \gamma_k \rightarrow \delta$, for δ an atomic type

A projection onto the i^{th} argument is possible only if $\delta = \beta$

Again, we want a truly incremental substitution that does not compromise any future decisions

Let H_i be a fresh variable of type $\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow \gamma_i$ for $i \leq m$

The i^{th} projection substitution for F, if it exists, is

$$\lambda x_1\ \ldots\ \lambda x_n\ (x_i\ (H_1\ x_1\ \ldots\ x_n)\ \ldots\ (H_k\ x_1\ \ldots\ x_n))$$

with the proviso that we introduce existential quantifiers for H_i adjacent to the one for F

Iteration of Simplification and Substitution Phases

The application of imitation or projection substitutions can produce new rigid-rigid pairs, thereby leading to an iteration

The iteration process can be organized into a structure called a *matching tree*, e.g. consider $\forall g_{\ i\to i\to i} \forall a_i\ \exists F_{i\to i}\ (F\ a) = (g\ a\ a)$

Some properties of matching trees

- simple types ensure the tree is finitely branching
  Note, though, that the tree structure is sensitive to typing
- "terminated" branches correspond to failure or (at most) flexible-flexible unification problems
- Some branches can go on for ever; for example, consider the imitation substitution $\lambda x\ u\ (H\ x)$ for F for

  $$\forall u_{i\to i}\ \exists F_{i\to i}\ \forall w_i\ [u\ (F\ w) = F\ (u\ w)]$$

  This yields a problem that has an identical structure

Properties of Flexible-Flexible Unification Problems

- Providing a unifier for such problems is easy
  A pair of the form

  $$(F\ t_1\ \ldots\ t_n) = (G\ s_1\ \ldots\ s_m)$$

  can be solved by picking a canonical variable of base type and projecting F and G onto that
- Providing a solution to such a problem requires showing habitation of the base type but this is decidable
- Enumeration of *all* unifiers can, however, be unstructured
  In fact, this part of the search cannot even be guaranteed to be non-redundant

Interestingly, solving flexible-flexible pairs becomes much more structured in the setting of the L_λ language

In fact, in this setting, there are even *most general* unifiers

Higher-Order Pattern Unification

Unification problems that arise in the L_λ setting are called higher-order pattern unification problems

We will observe that under the L_λ restrictions

- at most one branch in the matching tree can lead to success
- the flexible-flexible problems have a most general unifier
- With an additional "occurs-check" when processing flexible-rigid pairs, termination can be assured

Thus, higher-order pattern unification is decidable and admits the MGU property

The Matching Tree in the HOPU Setting

Flexible-rigid pairs in this context have the form

$$(F\ c_1\ \ldots c_n) = (c\ t_1\ \ldots t_m)$$

where $c_1, \ldots, c_n$ are distinct variables quantified universally within the scope of the quantifier for F

Consider, then, the possibilities for c

- c is quantified within the scope of the quantifier for F
  Thus
  - An imitation substitution cannot exist
  - At most one of the projections will be successful
- c is quantified outside the scope of the quantifier for F
  In this case only the imitation substitution will be applicable

It follows easily that there can be at most one success branch in the tree

Flexible-Flexible Problems in the HOPU Setting

One case of a flexible-flexible equation to consider is

$$(F\ c_1\ \ldots c_n) = (F\ d_1\ \ldots d_m)$$

where $c_1, \ldots, c_n$ and $d_1, \ldots, d_m$ are distinct variables quantified universally within the scope of the quantifier for F

Here the *same* substitution will be applied to the two sides

Thus, a c or d variable can appear in a common instance of the two terms only if $c_i = d_i$

Let $a_1, \ldots, a_k$ be a listing of the cs such that $c_i = d_i$

Then an MGU for this problem is

$$\{F \mapsto \lambda c_1\ \ldots \lambda c_n\ H\ a_i\ \ldots\ a_k\}$$

where H is a new existentially quantified variable

Flexible-Flexible HOPU Problems (Continued)

The other case of flexible-flexible equations to consider is

$$(F\ c_1\ \ldots c_n) = (G\ d_1\ \ldots d_m)$$

where $c_1, \ldots, c_n$ ($d_1, \ldots, d_m$) are distinct variables quantified universally within the scope of the quantifier for F (resp G)

Here a c or d variable can appear in a common instance of the two sides only if it appears *both* in the cs and in the ds

Let $a_1, \ldots, a_k$ be a listing of such variables

An MGU for the problem then is the substitution

$$\{\ \begin{array}{l} F \mapsto \lambda c_1\ \ldots \lambda c_n\ (H\ a_1\ \ldots\ a_k), \\ G \mapsto \lambda d_1\ \ldots \lambda d_m\ (H\ a_1\ \ldots\ a_k)\ \ \} \end{array}$$

where H is a new existentially quantified variable

Occurs-Check and Termination

Because substitutions produce only β_0-reductions, there is hope that the the matching tree will be finite

However, loops can still exist because of the flexible head can appear in the rigid term in a flexible-rigid pair, e.g. consider

$$\forall f_{i \to i}\ \exists X_i\ X = (f\ X)$$

To overcome this problem, we can include an "occurs-check" in the processing of flexible-rigid pairs

Fail if the flexible head appears in the rigid term

Note that this simple test *does not* work for the general higher-order unification setting

To illustrate the algorithm, consider the following problems

$$\forall f\ \forall g\ \exists U\ \exists V\ \forall w\ \forall x\ \forall y\ [(f\ (U\ x\ y)) = (f\ (g\ (V\ y\ w)))]$$
$$\forall f\ \forall g\ \exists U\ \exists V\ \forall w\ \forall x\ \forall y\ [(U\ x\ y) = (g\ (U\ y\ w))]$$
$$\forall f\ \forall g\ \exists U\ \exists V\ \forall w\ \forall x\ \forall y\ [(U\ x\ y) = (g\ w)]$$

Some Additional Observations about HOPU

- The incremental generation of substitutions leads to structure traversal that is highly redundant

  Instead, we can factor the process into two major steps
  - Simplification, that walks over the top-level rigid structure to at most leave flex-rigid and flex-flex pairs
  - Variable elimination, that takes a variable and a vector of arguments to try to generate a complete substitution
- In this form, the algorithm is similar to the first-order one except for a more involved variable elimination

  In fact, a nearly linear algorithm can be provided
- Preprocessing via raising can be practically costly

  Raising can be delayed and given an "on-the-fly" treatment

All the implementations discussed in this course use the first and third ideas