Unifiers and a Unification Algorithm

CS 440: Programming Languages and Translators, Spring 2020

Introduction

- We've seen that polymorphic typechecking uses type variables and instantiation (i.e., substitution with a type), and it requires unification: finding possibly many substitutions that can be done simultaneously to get various resulting types to match.

1. Unifiers

- Unification involves locating equalities between similar symbolic terms by finding a substitution that takes the terms and produces the same result. **Example**: If we want to unify the terms `f(X, 3)` and `f(5, Y)`, applying `[X ↦ 5, Y ↦ 3]` to both terms yields `f(5, 3)`: `f(X, 3) [X ↦ 5, Y ↦ 3] ≡ f(5, 3) ≡ f(5, Y) [X ↦ 5, Y ↦ 3]`. We say that `[X ↦ 5, Y ↦ 3]` is a **unifier** — a **solution** to the **equation** `f(X, 3) ≡ f(5, Y)`.
- An example of an equation that has no solution is `g(X, X) ≡ g(8, z)` because to solve it we need `X ↦ 8` and `X ↦ z` simultaneously, which would only work if `8 ≡ z`. Named constant identifiers like `z` don't get substituted for, so `8 ≢ z`.
- On the other hand, `g(X, X) ≡ g(8, Y)` does have a solution: `[X ↦ 8, Y ↦ 8]`.
  - The substitution `[X ↦ 8, Y ↦ X]` also works, but eventually, for reasons discussed later, we'll look for substitutions where the new terms (on the right of the ↦ arrows) doesn't involve any of the variables being substituted for (on the left of the ↦ arrows).
  - This will let us get to the most concrete substituted equations more quickly. E.g., if we apply `[X ↦ 8, Y ↦ X]` to `g(X, X) ≡ g(8, Y)`, we get `g(8, 8) ≡ g(8, X)`. We have to apply `[X ↦ 8, Y ↦ X]` another time to get to `g(8, 8) ≡ g(8, 8)`.
  - With this restriction, by the way, ordering the substitutions in parallel or sequentially doesn't change the result. E.g., for all terms t, t`[X ↦ 8, Y ↦ 8]` ≡ t`[X ↦ 8][Y ↦ 8]`.
- The restriction that substituted terms can't use any of the variables being substituted for also eliminates an infinite recursion problem. Allowing `[X ↦ f(X)]` and applying it to a term repeatedly until the result stabilizes doesn't work because `X` would have to become the infinitely long term `f(f(f(...)))`.

- **Definition**: **Unification** is the process of solving syntactic equations between terms by finding one substitution of variables to terms that, when applied to both sides of the equation, produces terms that are syntactically equal. Our terms include variables, constants, and function and operator terms $f(t_1, t_2, \ldots t_n)$ and t_1 *op* t_2, and **unification equations** are written as $t_1 \equiv t_2$ (the meaning of ≡ is becoming more general).
  - We'll use the simplest kind of unification. In **first-order unification**, function names and operators can't be variables; they have to be constants. Also, the number of arguments for a function is fixed.
  - Unification is a syntactic process, so we ignore semantic properties. E.g., `plus(pi, 2)` is ≢ `plus(2, pi)`.

2. *Most General Unifier*

- **Most general unifier (mgu).** Syntactic equations can have multiple solutions: E.g., `f(X)` ≡ `f(Y)` is solved by [`X` ↦ `0`, `Y` ↦ `0`] and by [`X` ↦ `1`, `Y` ↦ `1`], and so on, but both of these solutions are less general than [`X` ↦ `Y`]. In fact, all solutions are instances of this substitution, so we say that [`X` ↦ `Y`] is the **most general unifier (mgu)** of the equation `f(X)` ≡ `f(Y)`. The mgu is unique "up to renaming": [`Y` ↦ `X`] is also an mgu, but all we did was swap the names `X` and `Y`. If we ignore renaming, then we can say that [`X` ↦ `Y`] is "the" mgu. To be more specific about what "most general" means, we need some definitions.
- **Definition (Generalization and Specialization of Terms)**: Given terms t_1, t_2 and substitution σ, if t_1 σ ≡ t_2, then t_1 is **more general than** t_2; equivalently, t_2 is **more special than** t_1.
  - If a substitution σ just maps variables to variables, then t and t σ are **renamings** of each other, and each is simultaneously more general and more special than the other. (Basically, renamings don't affect generality.)
- However, not all maps of variables to variables are renaming operations. For example, [`X` ↦ `Z`, `Y` ↦ `Z`] maps `f(X, Y)` to `f(Z, Z)`, so `f(X, Y)` is more general than `f(Z, Z)`, but there isn't any substitution that takes `f(Z, Z)` to `f(X, Y)`, so `f(X, Y)` is not just more general than `f(Z, Z)`, it's *strictly* more general (and `f(Z, Z)` is strictly more specific than `f(X, Y)`).
- **Definition (Generalization and Specialization of Substitutions):** Given substitutions σ and τ, if for every term t, (t σ) is more general than (t τ), then σ is **more general than** τ; equivalently τ is **more special than** σ. We also say σ **subsumes** τ and τ is **subsumed by** σ.
- **Example**: [`X` ↦ `f(Y)`] is more general than [`X` ↦ `f(g(Z))`].
- **Example**: Let t_1 ≡ t [`X` ↦ `Z`] and t_2 ≡ t [`X` ↦ `Z`, `Y` ↦ `Z`], where t is an arbitrary term. We know t_1 is more general than t_2 because there's a substitution that takes t_1 to t_2, namely [`Y` ↦ `Z`]. Since t was any term, we know [`X` ↦ `Z`] is more general than [`X` ↦ `Z`, `Y` ↦ `Z`].

3. *Equations and Unification Problems*

- **Definitions**: A **unification equation** is a pair of terms usually written t ≡ u, but we can also write (t, u). The equation represents the statement "t and t can be unified", or "t σ ≡ u σ" where σ is unknown. (From the context, we know that ≡ specifically means syntactic equality here.)
  - A **unification problem** is a set of equations {t_1 ≡ u_1, t_2 ≡ u_2, …, t_n ≡ u_n}.
  - A **solution** to this unification problem is a substitution σ that simultaneously solves the equations (t_1 σ ≡ u_1 σ), (t_2 σ ≡ u_2 σ), …, (t_n σ ≡ u_n σ)[1]. We also say σ is a **unifier** of/for the problem.
  - We say σ is a **most general unifier (mgu)** for the problem if it is a more general substitution than all other unifiers for the problem. All mgu's are renamings of each other.

[1] The parentheses are optional.

4. A Unification Algorithm

- Let problem $P = \{t_1 \equiv u_1, t_2 \equiv u_2, \ldots, t_n \equiv u_n\}$. I.e., P is a set of unification equations. If, say, t_1 is just a variable X_1, then the problem $t_1 \equiv u_1$ is $X_1 \equiv u_1$, which is solved by $[X_1 \mapsto u_1]$.
- The goal of the algorithm is to take P and develop a list of substitutions $[X_1 \mapsto s_1, X_2 \mapsto s_2, \ldots, X_m \mapsto s_m]$ that unify all the original equations. The substitution list is initially empty; as it grows, P gets smaller.
- The algorithm works by repeatedly removing an arbitrary equation $t \equiv u$ from P and using it to simplify the problem. There some cases that are solvable immediately: $X \equiv X$, $X \equiv Y$, and $X \equiv u$. There is a case that requires recursion, $f(t_1, \ldots, v_n) \equiv g(u_1, \ldots, u_m)$. (And there are cases that immediately cause failure.)

To solve a problem set *P:*

let $S = \varnothing$ // S is the set of solution equations
while P has an equation $(t \equiv u)$
. *let* $P' \leftarrow P$ less the member $(t \equiv u)$ in
. *if* one or both of t and u are variables *then*
. . *let* X be the variable and s be the non-variable (or other variable) *in*
. . . *if* s is also a variable and $X \equiv s$ *then skip*
. . . *else* // add $[X \mapsto s]$ as a substitution but apply it to P' and S first
. . . . *set* $P' \leftarrow P'\,[X \mapsto s]$ // see note below
. . . . *set* $S \leftarrow S\,[X \mapsto s] \cup \{[X \mapsto s]\}$ // see note below
. *else if* t is a constant or identifier *then*
if $u \equiv$ the same constant or identifier, *then skip else fail*
. *else if* $t \equiv f(v_1, v_2, \ldots, v_n)$ *then*
. . *if* u is a function call $g(w_1, w_2, \ldots, w_m)$ where $f \equiv g$ and $m = n$
. . *then* add the equations $(v_1 \equiv w_1), (v_2 \equiv w_2), \ldots, (v_n \equiv w_n)$ to P'
. . *else fail*
. Go on to the next iteration, with P' for P
end

- Note: Before adding a new substitution $[X \mapsto u]$ to S, we apply it to P' and S so that the X will no longer appear in P' and S. This avoids building an S that contains chains of substitutions like $[\mathtt{Y} \mapsto \mathtt{X}]$, $[\mathtt{X} \mapsto \mathtt{8}]$.
  - $P'\,\sigma$ = the set of equations $(v\sigma \equiv w\sigma)$ where $(v \equiv w) \in P'$.
  - $S'\,\sigma$ = the set of substitutions $[Y \mapsto e\sigma]$ where $[Y \mapsto e] \in S$.

5. Examples of Unification

- Here's a table with some examples of unification problems and their solutions (if any).

Problem	*Solution, if any*
{X ≡ Y, X ≡ 3}	[Y ↦ 3, X ↦ 3] (not [Y ↦ 3, X ↦ Y])
{X ≡ 1, X ≡ 3}	Fails: Tries to unify 1 and 3
{f(a, Y) ≡ f(X, b), c ≡ Z}	[Z ↦ c, Y ↦ b, X ↦ a]
{f(X) ≡ g(Y)}	Fails: different function names
{f(X, Y) ≡ f(X)}	Fails: different # of function arguments
{f(f(f(f(a, Z), Y), X), W) ≡ f(W, f(X, f(Y, f(Z, a))))} (Found this one in Wikipedia)	[Z ↦ a, Y ↦ f(a, a), X ↦ f(f(a, a), f(a, a)), W ↦ f(f(f(a, a), f(a, a)), f(f(a, a), f(a, a)))]

- Note: Since problems and solutions are sets, reordering elements doesn't make a problem or solution different. Similarly, flipping an equation $t \equiv u$ to $u \equiv t$ doesn't change a problem, nor does repeating an equation.

6. Traces of Unification Algorithm

- Below, each row of a table represents one iteration of the unification algorithm loop.

Problems P	*Equation*	*Problems P′*	*Action*	*New Substitutions S*
{X ≡ Y, X ≡ 3}	X ≡ Y	{X ≡ 3}	$P \leftarrow P'\sigma$ and $S \leftarrow S\,\sigma \cup \{\sigma\}$ where σ = [X ↦ Y]	[X ↦ Y]
{Y ≡ 3}	Y ≡ 3	∅	$P \leftarrow P'\sigma$ and $S \leftarrow S\,\sigma \cup \{\sigma\}$ where σ = [Y ↦ 3]	[Y ↦ 3, X ↦ 3]

Problems P	*Equation*	*Problems P′*	*Action*	*New Substitutions S*
{X ≡ 1, X ≡ 3}	X ≡ 1	{X ≡ 3}	$P \leftarrow P'\sigma$ and $S \leftarrow S \cup \{\sigma\}$ where σ = [X ↦ 1]	[X ↦ 1]
{1 ≡ 3}	1 ≡ 3	∅	Unification fails bec. 1 ≢ 3	

Problems P	*Equation*	*Problems P′*	*Action*	*New Substitutions S*
{f(X,X,2) ≡ f(5,Y,Z)}	f(X,X,2) ≡ f(5,Y,Z)	∅	$P \leftarrow P' \cup \{X \equiv 5, X \equiv Y, 2 \equiv Z\}$	∅
{X ≡ 5, X ≡ Y, 2 ≡ Z}	X ≡ 5	{X ≡ Y, 2 ≡ Z}	$P \leftarrow P'\sigma$ and $S \leftarrow S \cup \{\sigma\}$ where $\sigma = [X \mapsto 5]$	[X ↦ 5]
{5 ≡ Y, 2 ≡ Z}	5 ≡ Y	{2 ≡ Z}	$P \leftarrow P' \cup \{[Y \equiv 5]\}$	[X ↦ 5]
{Y ≡ 5, 2 ≡ Z}	Y ≡ 5	{2 ≡ Z}	$P \leftarrow P'\sigma$ and $S \leftarrow S \cup \{\sigma\}$ where $\sigma = [Y \mapsto 5]$	[Y ↦ 5, X ↦ 5]
{2 ≡ Z}	2 ≡ Z	∅	$P \leftarrow P'\sigma$ and $S \leftarrow S \cup \{\sigma\}$ where $\sigma = [Z \equiv 2]$	[Y ↦ 5, X ↦ 5]
{Z ≡ 2}	Z ≡ 2	∅	$P \leftarrow P'[Z \mapsto 2]$ $S \leftarrow S\,[Z \mapsto 2] \cup \{[Z \mapsto 2]\}$	[Z ↦ 2, Y ↦ 5, X ↦ 5]

Activity Questions for Lecture 16

Lecture 16: Unification

1. What is a unification equation and how do we write them? What is a unification problem and how do we write one? What does it mean to solve an equation or problem? What is a "unifier"?

2. How do *most general unifiers* (mgu) differ from non-mgu's? Why do we say "an" mgu versus "the" mgu?

3. For each of the following, give a brief explanation as to why the unification problem has no solution.
   a. {`X` ≡ `Y`, `X` ≡ `Y+Y`}
   b. {`f(a, b, Y)` ≡ `g(X, b)`}
   c. {`f(X, Y, Z)` ≡ `f(X, X, Z)`, `f(X, Y, Z)` ≡ `f(Y, Z, Z)`, `f(A, B, B)` ≡ `f(1, 2, B)`}

4. Solve each of the following unification problems. Look for the most general unifier.
   a. {`X` ≡ `Y`, `Y` ≡ `Z`, `Z` ≡ `X`}
   b. {`f(X, Y, Z)` ≡ `f(X, X, Z)`, `f(X, Y, Z)` ≡ `f(Y, Z, Z)`, `f(A, B, B)` ≡ `f(x, x, x)`}
   c. {`T0` ≡ `n(a, T1, T2)`, `n(a, T1, T2)` ≡ `n(a, n(b, c, Z), n(d, T3, T4))`,
      `n(d, T3, n(g, h, Z))` ≡ `n(d, n(e, Z, f), T4)` }

Solutions to Activity Questions for Lecture 16

Lecture 16: Unifiers and a Unification Algorithm

1. A unification equation is written $s \equiv t$ where s and t are terms. A unification problem is a set of unification equations and is written $\{s_1 \equiv t_1, s_2 \equiv t_2, \ldots\}$. Solving an equation $s \equiv t$ means finding a substitution σ that sends them to the same term: $s\,\sigma \equiv t\,\sigma$ (they have to be syntactically equal). Solving a problem means finding a substitution that solves all the equations simultaneously: $s_1\,\sigma \equiv t_1\,\sigma$, $s_2\,\sigma \equiv t_2\,\sigma$, …. A unifier is such a substitution.

2. A most general unifier is one that encompasses all possible unifiers: If $[X_1 \mapsto u_1, X_2 \mapsto u_2, \ldots]$ is an mgu, then all other unifiers have the form $[X_1 \mapsto u_1\,\tau, X_2 \mapsto u_2\,\tau, \ldots]$ for all τ. Mgu's are unique "up to renaming": two mgu's differ only in the names of the variables they produce. E.g., [X ↦ Y, Z ↦ Y] and [X ↦ Z, Y ↦ Z] are renamings of each other.

3. (Why do unifications fail?)
   a. With {`X` ≡ `Y`, `X` ≡ `Y+Y`}, there's no way to unify `Y` and `Y+Y`.
   b. With {`f(a, b, Y)` ≡ `g(X, b)`}, the function names don't match; neither do the number of arguments.
   c. With {`f(X, Y, Z)` ≡ `f(X, X,Z)`, `f(X, Y, Z)` ≡ `f(Y, Z,Z)`, `f(A, B, B)` ≡ `f(1, 2,B)`}, the first equation tells us `X` ≡ `Y` and the second equation tells us `Y` ≡ `Z`. (This much is solvable, with [`X` ↦ `Z`][`Y` ↦ `Z`].) The third equation tells us `A` ≡ `1`, `B` ≡ `2`, which is fine, but there's no way to unify `f(X, X,Z)` from equation 1 with `f(1, 2,B)`, so overall, unification fails.

4. (Solve unification problems)
   a. From {`X` ≡ `Y`, `Y` ≡ `Z`, `Z` ≡ `X`} we get [`X` ↦ `Y`], so the problem reduces to {`Y` ≡ `Z`, `Z` ≡ `X`}[`X` ↦ `Y`], which ≡ {`Y` ≡ `Z`, `Z` ≡ `Y`}.

      This gives us [`Y` ↦ `Z`], reducing the problem to {`Z` ≡ `Y`}[`Y` ↦ `Z`] which equals {`Z` ≡ `Y`[`Y` ↦ `Z`]} which equals {`Z` ≡ `Z`} and changes our partial solution from [`X` ↦ `Y`] to [`X` ↦ `Y`[`Y` ↦ `Z`]], which ≡ [`X` ↦ `Z`].

      The problem {`Z` ≡ `Z`} is solvable without adding any substitution. The solution is [`X` ↦ `Z`, `Y` ↦ `Z`].

   b. From {`f(X, Y, Z)` ≡ `f(X, X,Z)`, `f(X, Y, Z)` ≡ `f(Y, Z,Z)`, `f(A, B, B)` ≡ `f(x, x, x)`},
      we reduce to {`X` ≡ `X`, `Y` ≡ `X`, `Z` ≡ `Z`, `f(X, Y, Z)` ≡ `f(Y, Z,Z)`, `f(A, B, B)` ≡ `f(x, x, x)`}.
      Eventually this gives us [`Y` ↦ `X`], which modifies the problem to

      {`f(X, X, Z)` ≡ `f(X, Z, Z)`, `f(A, B, B)` ≡ `f(x, x, x)`}

      which eventually leads to adding [`X` ↦ `Z`] to [`Y` ↦ `X`] and {`f(A, B, B)` ≡ `f(x, x, x)`}
      to get [`Y` ↦ `Z`, `X` ↦ `Z`] and {`f(A, B, B)` ≡ `f(x, x, x)`}, which eventually leads to the final solution, [`Y` ↦ `Z`, `X` ↦ `Z`, `A` ↦ `x`, `B` ↦ `x`]. Note you might have thought that all five of `X`, `Y`, `Z`, `A`, and `B`

$\mapsto$ x, but the two equations don't share any variables, so they have individual solutions.

c. I'll leave you to work this one out, but

$$\{T0 \equiv n(a, T1, T2),$$
$$n(a, T1, T2) \equiv n(a, n(b, c, Z), n(d, T3, T4)),$$
$$n(d, T3, n(g, h, Z)) \equiv n(d, n(e, Z, f), T4)\ \}$$

ends up with

$$[T0 \mapsto n(a, n(b, c, Z), n(d, n(e, Z, f), n(g, h, Z))),$$
$$T1 \mapsto n(b, c, Z),$$
$$T2 \mapsto n(d, n(e, Z, f), n(g, h, Z)),$$
$$T3 \mapsto n(e, Z, f),$$
$$T4 \mapsto n(g, h, Z)]$$

(Hint: You can read these as trees if you like: *node*(*root value*, *left subtree*, *right subtree*) and Z means null pointer.)

