

## 1 The properties of unification

We begin today's lecture on the properties of the unification algorithm. To be specific, the equation we are reasoning about is  $\sigma(t_i) = \sigma(s_i)$ . We would like the function `unify` to give us the most general substitution when it returns a solution to a set of constraints. For instance, the  $\sigma$  that results from  $\sigma = \text{unify}\{t_i, s_i\} \Rightarrow \sigma(t_i) = \sigma(s_i)$  should be the most general substitution for that set of constraints. How do we formally state such an intuition?

$$\text{if } \sigma_1 \leq \sigma_2 \text{ then } \exists \sigma_3. \sigma_3 \circ \sigma_1 = \sigma_2$$

Before moving forward it would be helpful to define exactly what composition is. Composition can be defined as  $\sigma_1 \circ \sigma_2 \equiv \sigma_3$  such that  $\forall t. \sigma_3(t) = \sigma_1(\sigma_2(t))$ . When we compose substitutions from other substitutions, we are saying we can generate a new substitution from applying one to another. In other words,

$$\sigma_1 \circ \sigma_2 \equiv \sigma_1 @ [x_i = \sigma_1(t_i)] \text{ where } [x_i = t_i] = \sigma_2$$

For example, say we had  $\sigma_1 \leq \sigma_2$ . Then we could say  $\sigma_1 = [x = y]$  and  $\sigma_2 = [x=17, y=17]$ .  $\sigma_1$  is the more general substitution. Given the above substitutions, what is  $\sigma_3$ ?  $\sigma_3 = [y=17]$ , because when  $\sigma_1$  is applied all of the occurrences of  $x$  will be replaced by  $y$ .

As a homework problem, prove that the above sigmas prove that  $\exists \sigma_3. \sigma_3 \circ \sigma_1 = \sigma_2$

Now we move on to some more general properties of unification. We'd like to say that if the function `unify` returns a unifier, then that unifier will satisfy the set of constraints which `unify` took in as input. Second, we would like to say that if there does exist a unifier for a set of constraints, then the `unify` algorithm will produce some output in the form of a unifier. These two notions are captured by the following formalisms:

$$\sigma = \text{unify}\{t_i, s_i\} \Rightarrow (\sigma(t_i) = \sigma(s_i))$$

Conversely,

$$\exists \sigma. \sigma(t_i) = \sigma(s_i) \Rightarrow \exists \sigma'. \sigma' = \text{unify}\{t_i, s_i\}$$

Whats missing here? These two say unification works. Now we would like to say something about the best unifier that can be found. If unify returns a solution,  $\sigma$ , then unify returns the most general solution:

$$\forall \sigma'. \sigma'(t_i = \sigma'(s_i)) \Rightarrow \sigma \leq \sigma'$$

How do we prove this? It is important to remember that not every set of constraints has a most general unifier. More importantly, we need to remember the difference between a unifier and a substitution: A unifier is a substitution that satisfies all equations in a set of constraints.

The unification algorithm presented in chapter 22 of the text gives rise to the notion of a most general type. For instance, say we have the following let binding:

let  $x = (\lambda y. y)$  in  $xx$

Can we type this expression in our type system? No, because you cannot pass the same type to a type in our system. For this we need let-polymorphism.