4.6 Unification by transformation

Unification can be expressed as a repeated transformation of a set of equations until the solution stares you in the face. In fact, it is very reminiscent of solving systems of linear equations by Gaussian elimination, as in the following example:

$$\boxed{\begin{aligned} x + 3y &= 0 \\ 2x + 8y &= 2z \end{aligned}} \rightsquigarrow \boxed{\begin{aligned} x + 3y &= 0 \\ 2y &= 2z \end{aligned}} \rightsquigarrow \boxed{\begin{aligned} x + 3y &= 0 \\ y &= z \end{aligned}}$$

$$\rightsquigarrow \boxed{\begin{aligned} x + 3z &= 0 \\ y &= z \end{aligned}} \rightsquigarrow \boxed{\begin{aligned} x &= -3z \\ y &= z \end{aligned}}$$

We leave it to the reader to compare the individual steps of Gaussian elimination and the transformations below.

Definition 4.6.1 A unification problem $S = \{x_1 \stackrel{?}{=} t_1, \ldots, x_n \stackrel{?}{=} t_n\}$ is in **solved form** if the x_i are pairwise distinct variables, none of which occurs in any of the t_i. In this case we define

$$\vec{S} := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}.$$

Lemma 4.6.2 *If S is in solved form then $\sigma = \sigma\vec{S}$ for all $\sigma \in \mathcal{U}(S)$.*

Proof Let $S = \{x_1 =^? t_1, \ldots, x_n =^? t_n\}$. We show by case distinction that σ and $\sigma\vec{S}$ behave the same on all variables, i.e. $\forall x \in V.\ \sigma x = \sigma\vec{S}x$:

1. $x \in \{x_1, \ldots, x_n\}$, e.g. $x = x_k$: $\sigma x = \sigma t_k = \sigma\vec{S}x$ (because $\sigma \in \mathcal{U}(S)$).
2. $x \notin \{x_1, \ldots, x_n\}$: $\sigma x = \sigma\vec{S}x$ (because $\vec{S}x = x$). □

Lemma 4.6.3 *If S is in solved form then $\vec{S}$ is an idempotent mgu of S.*

Proof Idempotence follows directly from Lemma 4.5.7 because none of the x_i occurs in the t_i. For the same reason we have $\vec{S}x_i = t_i = \vec{S}t_i$, i.e. $\vec{S} \in \mathcal{U}(S)$. Finally, $\vec{S}$ is an mgu because $\vec{S} \lesssim \sigma$ for all $\sigma \in \mathcal{U}(S)$ by Lemma 4.6.2. □

Thus we know how to extract an idempotent mgu once we have reached a solved form. In order to get there we employ the following transformation rules:

Delete	$\{t =^? t\} \uplus S$	$\Longrightarrow$	S
Decompose	$\{f(\overline{t_n}) =^? f(\overline{u_n})\} \uplus S$	$\Longrightarrow$	$\{t_1 =^? u_1, \ldots, t_n =^? u_n\} \cup S$
Orient	$\{t =^? x\} \uplus S$	$\Longrightarrow$	$\{x =^? t\} \cup S$ if $t \notin V$
Eliminate	$\{x =^? t\} \uplus S$	$\Longrightarrow$	$\{x =^? t\} \cup \{x \mapsto t\}(S)$ if $x \in \mathcal{V}ar(S) - \mathcal{V}ar(t)$

The application of a substitution to S means its application to both sides of all equations in S. The symbol $\uplus$ denotes disjoint union. This enforces that the particular equation selected by the left-hand side is removed from set of equations (although Eliminate reinserts it). The individual rules are easy to grasp:

Delete deletes trivial equations.

Decompose replaces equations between terms by equations between their subterms.

Orient moves variables to the left-hand side.

Eliminate broadcasts solutions, thereby eliminating the solved variable in the remaining part of the problem.

Dropping the side conditions can cause looping. For example $x \in \mathcal{V}ar(t)$ in Eliminate has the following consequence:

$$\begin{aligned} \{x =^? f(x), \ldots x \ldots\} &\Longrightarrow \{x =^? f(x), \ldots f(x) \ldots\} \\ &\Longrightarrow \{x =^? f(x), \ldots f(f(x)) \ldots\} \Longrightarrow \cdots. \end{aligned}$$

Example 4.6.4 The following sequence of transformations illustrates the workings of the above rules:

$$\begin{array}{ll}
\{x =^? f(a),\ g(x,x) =^? g(x,y)\} & \Longrightarrow_{\mathsf{Eliminate}} \\
\{x =^? f(a),\ g(f(a),f(a)) =^? g(f(a),y)\} & \Longrightarrow_{\mathsf{Decompose}} \\
\{x =^? f(a),\ f(a) =^? f(a),\ f(a) =^? y\} & \Longrightarrow_{\mathsf{Delete}} \\
\{x =^? f(a),\ f(a) =^? y\} & \Longrightarrow_{\mathsf{Orient}} \\
\{x =^? f(a),\ y =^? f(a)\}. &
\end{array}$$

Note that the choice of rules is nondeterministic. We could start with Decompose instead of Eliminate. Would this lead to a different solved form?

The solved form we end up with yields an mgu $\{x \mapsto f(a), y \mapsto f(a)\}$ for our initial unification problem. This is not a coincidence. We claim that the following function *Unify* computes a most general unifier if one exists and fails otherwise.

$$\begin{array}{lll}
\mathit{Unify}(S) & = & \texttt{while there is some } T \text{ such that } S \Longrightarrow T \texttt{ do } S := T; \\
 & & \texttt{if } S \text{ is in solved form then return } \vec{S} \texttt{ else fail.}
\end{array}$$

Note that the above algorithm is nondeterministic: if there is more than one applicable transformation rule, e.g. $S \Longrightarrow T_1$ and $S \Longrightarrow T_2$, the algorithm may choose an *arbitrary* one. Termination of *Unify* thus depends on termination of $\Longrightarrow$. The latter is not completely trivial because Eliminate may increase the size of a unification problem, as Example 4.6.4 shows.

Lemma 4.6.5 *Unify terminates for all inputs.*

Proof We call a variable x **solved** if it occurs exactly once in S, namely on the left-hand side of some equation $x =^? t$ where $x \notin \mathcal{V}ar(t)$.

Termination of $\Longrightarrow$ is proved by a measure function that maps a unification problem S to a triple (n_1, n_2, n_3) of natural numbers such that

n_1 is the number of variables in S that are not solved,
n_2 is the size of S, i.e. $\sum_{(s=^?t)\in S}(|s| + |t|)$, and
n_3 is the number of equations $t =^? x$ in S.

The following table shows that each step decreases the triples w.r.t. the lexicographic order:

	n_1	n_2	n_3
Delete	$\geq$	$>$	
Decompose	$\geq$	$>$	
Orient	$\geq$	$=$	$>$
Eliminate	$>$		

The interpretation is obvious: Eliminate decreases n_1, which none of the other rules can increase. Delete and Decompose decrease n_2. Orient leaves n_2 unchanged but decreases n_3.

For example, the transformations sequence in Example 4.6.4 is mapped to $(2,9,0) >_{lex} (1,12,0) >_{lex} (1,10,1) >_{lex} (1,6,1) >_{lex} (0,6,0)$. □

The key property of $\Longrightarrow$ is preservation of unifiers:

Lemma 4.6.6 *If $S \Longrightarrow T$ then $\mathcal{U}(S) = \mathcal{U}(T)$.*

Proof For Delete, Decompose and Orient this is obvious.

For Eliminate let $\theta := \{x \mapsto t\}$. Applying Lemma 4.6.2 to $x =^? t$, which is in solved form, we get that $\sigma = \sigma\theta$ if $\sigma x = \sigma t$. Thus we conclude that

$$\begin{aligned} \sigma \in \mathcal{U}(\{x =^? t\} \uplus S) &\Leftrightarrow \sigma x = \sigma t \wedge \sigma \in \mathcal{U}(S) \\ &\Leftrightarrow \sigma x = \sigma t \wedge \sigma\theta \in \mathcal{U}(S) \\ &\Leftrightarrow \sigma x = \sigma t \wedge \sigma \in \mathcal{U}(\theta S) \\ &\Leftrightarrow \sigma \in \mathcal{U}(\{x =^? t\} \cup \theta S). \end{aligned}$$ □

The following lemma expresses soundness of *Unify* and follows directly from Lemmas 4.6.3 and 4.6.6 above.

Lemma 4.6.7 *If Unify(S) returns a substitution σ then σ is an idempotent mgu of S.*

The completeness proof requires two fundamental properties of terms:

Lemma 4.6.8 *An equation $f(\overline{s_m}) =^? g(\overline{t_n})$, where $f \neq g$, has no solution.*

Proof $\sigma(f(\overline{s_m})) = f(\overline{\sigma(s_m)}) \neq g(\overline{\sigma(t_n)}) = \sigma(g(\overline{t_n}))$. □

Lemma 4.6.9 *An equation $x =^? t$, where $x \in \mathcal{V}ar(t)$ and $x \neq t$, has no solution.*

Proof If $x \neq t$ then t is of the form $f(\overline{t_n})$ with $x \in \mathcal{V}ar(t_i)$ for some i. Hence $\sigma(x)$ and $\sigma(t)$ cannot be identical because $|\sigma(x)| \leq |\sigma(t_i)| < |\sigma(t)|$. □

Now we obtain completeness of *Unify*, i.e. every solvable system is solved:

Lemma 4.6.10 *If S is solvable, Unify(S) does not fail.*

Proof By Lemma 4.6.6 it suffices to show that if S is solvable and in normal form w.r.t. $\Longrightarrow$, then S is in solved form.

S cannot contain equations of the form $f(\ldots) =^? f(\ldots)$ (because of Decompose), $f(\ldots) =^? g(\ldots)$ (because of Lemma 4.6.8), $x =^? x$ (because of Delete), and $t =^? x$ where $t \notin V$ (because of Orient). Hence all equations in S are of the form $x =^? t$ where $x \notin \mathcal{V}ar(t)$ (because of Lemma 4.6.9).

Because of Eliminate, x cannot occur twice in S. Hence S is in solved form. □

Theorem 4.5.8 is now a direct consequence of soundness, completeness and termination of *Unify*.

Detecting unsolvability with the above rules for $\Longrightarrow$ can be a lengthy affair because one has to compute a normal form first. Therefore we introduce a special unification problem $\bot$ that has no solution, and add the two rules

Clash	$\{f(\overline{t_m}) =^? g(\overline{u_n})\} \uplus S$	$\Longrightarrow$	$\bot$	if $f \neq g$
Occurs-Check	$\{x =^? t\} \uplus S$	$\Longrightarrow$	$\bot$	if $x \in \mathcal{V}(t)$ and $x \neq t$

which are immediately justified by Lemmas 4.6.8 and 4.6.9. Here is a very simple example of the behaviour of the extended set of rules:

$$\{f(x,x) =^? f(y,g(y))\} \Longrightarrow \{x =^? y,\ x =^? g(y)\} \Longrightarrow \{x =^? y,\ y =^? g(y)\} \Longrightarrow \bot.$$

Since $\bot$ is not in solved form, *Unify* fails on problems with normal form $\bot$.

The complexity of *Unify* is (at least) exponential in both time and space.

Example 4.6.11 It is easy to see that the unification problem

$$\{x_1 =^? f(x_0,x_0),\ x_2 =^? f(x_1,x_1),\ \ldots,\ x_n =^? f(x_{n-1},x_{n-1})\}$$

has the idempotent mgu

$$\{x_1 \mapsto f(x_0,x_0), x_2 \mapsto f(f(x_0,x_0),f(x_0,x_0)),\ldots\}$$

which maps x_i to a complete binary tree of height i. Thus the size of every mgu of this example is exponential in the size of the input (because mgus are equal modulo renaming).

The above unification problem can also be obtained by unifying only two terms containing just variables and the binary f:

$$\begin{array}{lcl} s_n(x) & = & f(x_1, \quad f(x_2, \quad f(\ldots,x_n \quad)\ldots)), \\ t_n(x) & = & f(f(x_0,x_0),\ f(f(x_1,x_1),\ f(\ldots,f(x_{n-1},x_{n-1}))\ldots)). \end{array}$$

Note that the names of the x_i on the rhs depend directly on the x on the lhs. This becomes relevant in a later example where we need different instances of s_n and t_n with different variable names x and y.

Computing the mgu in this example requires exponential space because of copying. If the underlying implementation is based on graphs and sharing, as opposed to trees, linear space complexity is obtained. We come back to this implementation technique in Section 4.8.

Before we concentrate on algorithmic issues, let us briefly explore the connection between unification and matching. Recall that (syntactic) matching is the problem of finding a substitution σ such that $\sigma(s) = t$. We denote the matching problem by $s \lesssim^? t$ and call σ a solution of $s \lesssim^? t$ or a **matcher** of s and t, in which case we say that s **matches** t. The extension to finite sets of matching problems is analogous to unification.

Note that all solutions of $s \lesssim^? t$ coincide on $\mathcal{V}ar(s)$ (see Exercise 4.14). Thus a matcher is unique, as far as s and t are concerned.

We can easily reduce matching to unification: simply regard all variables in t as constants, for example by introducing a new constant c_x for each variable x.

Example 4.6.12 The matching problem $f(x, y) \lesssim^? f(g(z), x)$ becomes the unification problem $f(x, y) =^? f(g(c_z), c_x)$. The unifier $\{x \mapsto g(c_z), y \mapsto c_x\}$ becomes the matcher $\{x \mapsto g(z), y \mapsto x\}$.

In many applications of matching we may assume that t is ground. In that case unification and matching trivially coincide. From a complexity point of view, unification and matching do not differ very much either: both can be implemented in linear time. However, linear implementations of matching are quite straightforward (see Exercise 4.24), whereas linear unification requires sophisticated data structures and will occupy us for much of the rest of this chapter. Therefore matching should be implemented separately from unification if efficiency is an issue.

Exercises

4.16 Let S and T be unification problems. Show that if σ is an mgu of S and θ an mgu of $\sigma(T)$ then $\theta\sigma$ is an mgu of $S \cup T$.

4.17 Show that after the addition of the rules Clash and Occurs-Check, the second line in function *Unify* can be rephrased as follows:

$$\texttt{if } S = \bot \texttt{ then fail else return } \vec{S}.$$

4.18 Check if the following unification/matching problems are solvable:

(a) $f(x, y) \;=^? / \lesssim^?\; f(h(a), x)$;
(b) $f(x, y) \;=^? / \lesssim^?\; f(h(x), x)$;
(c) $f(x, b) \;=^? / \lesssim^?\; f(h(y), z)$;
(d) $f(x, x) \;=^? / \lesssim^?\; f(h(y), y)$.

4.19 Modify the transformation rules for unification (including Occurs-Check and Clash) such that they directly solve the matching problem (rather than first replacing all variables on the right-hand sides by

constants). Allow for variables on both sides and detect unsolvability as early as possible.

4.20 Does every matching problem have an idempotent solution? Can you find a sufficient condition for the existence of an idempotent solution?