First-Order Unification by Structural Recursion Correctness Proof

Conor McBride
Department of Computer Science
University of Durham
c.t.mcbride@durham.ac.uk

We will shortly seek to prove that the substitutions computed by $\mathbf{mgu}_m\ s\ t$ have certain properties—that they unify s and t and that they are the most general substitutions which do so. Let us build some apparatus to help us.

1 Describing Properties of Substitutions

When we compute a unifier, we know in advance its *source* number of variables, but the size of the *target* variable set may vary. Let us define what it means to be a *property* of substitutions with m source variables:

$$\text{let} \quad \frac{m \ :\ \mathbb{N}}{\mathbf{Property}\ m \ :\ \mathsf{Type}}$$

$$\mathbf{Property}\ m \ \mapsto\ \forall n{:}\mathbb{N}.\ (\mathsf{Fin}\ m \ \to\ \mathsf{Term}\ n) \ \to\ \mathsf{Type}$$

When stating that a property P holds for a substitution f, I shall typically write $P\ f$, suppressing the target number n which is easily inferred from f's type.

Being a unifier is such a property

$$\text{let} \quad \frac{s, t \ :\ \mathsf{Term}\ m}{\mathbf{Unifies}\ s\ t \ :\ \mathbf{Property}\ m} \qquad \mathbf{Unifies}\ s\ t\ f \ \mapsto\ f \lozenge s = f \lozenge t$$

We may define pointwise conjunction by exploiting the cartesian product:

$$\text{let} \quad \frac{P, Q \ :\ \mathbf{Property}\ m}{P \wedge Q \ :\ \mathbf{Property}\ m} \qquad (P \wedge Q)\, f \ \mapsto\ P\, f \times Q\, f$$

We may also define *equivalence* of properties:

$$\text{let} \quad \frac{P, Q \ :\ \mathbf{Property}\ m}{P \Leftrightarrow Q \ :\ \mathsf{Type}}$$

$$P \Leftrightarrow Q \ \mapsto\ \forall n, f.\ (P\, f \ \to\ Q\, f) \times (Q\, f \ \to\ P\, f)$$

In particular,

$$\text{fact} \quad \begin{array}{l} \mathbf{Unifies}\; s\; t \Leftrightarrow \mathbf{Unifies}\; t\; s \\ \mathbf{Unifies}\; (s_1\; \mathsf{fork}\; s_2)\; (t_1\; \mathsf{fork}\; t_2) \Leftrightarrow \mathbf{Unifies}\; s_1\; t_1 \wedge \mathbf{Unifies}\; s_2\; t_2 \end{array}$$

Further, we may explain when a property fails to hold, using the empty type, $\perp$:

$$\text{let} \quad \frac{P \;:\; \mathbf{Property}\; m}{\mathbf{Nothing}\; P \;:\; \mathsf{Type}} \qquad \mathbf{Nothing}\; P \;\mapsto\; \forall n, f.\, P\, f \;\to\; \perp$$

$$\text{fact} \quad \frac{P \Leftrightarrow Q \qquad \mathbf{Nothing}\; P}{\mathbf{Nothing}\; Q}$$

As we intend to work with accumulated substitutions, we need to describe the property of *being an extension* to some f which satisfies some P:

$$\text{let} \quad \frac{P \;:\; \mathbf{Property}\; m \qquad f \;:\; \mathsf{Fin}\; m \;\to\; \mathsf{Term}\; n}{P[\_ \diamond f] \;:\; \mathbf{Property}\; n} \qquad P[\_ \diamond f]\; g \;\mapsto\; P(g \diamond f)$$

Any f satisfying P extends ι. Also, if P always fails, it fails for any accumulator.

$$\text{fact} \quad P \Leftrightarrow P[\_ \diamond \iota] \qquad \frac{\mathbf{Nothing}\; P}{\mathbf{Nothing}\; P[\_ \diamond f]}$$

Iterating the accumulator construct composes the accumulators, so we may apply part of the accumulator whenever we like:

$$\text{fact} \quad P[\_ \diamond g][\_ \diamond f] \Leftrightarrow P[\_ \diamond (f \diamond g)]$$

$$(\mathbf{Unifies}\; s\; t)[\_ \diamond (f \diamond g)] \Leftrightarrow (\mathbf{Unifies}\; (g \,\langle\!\!\!|\, s)\; (g \,\langle\!\!\!|\, t))[\_ \diamond f]$$

2 Optimizing Properties

In order to talk about ‘most general unifiers’, we need a notion of ordering on substitutions. Composition induces an obvious preorder with a top element, preserved by precomposition

$$\text{let} \quad \frac{f \;:\; \mathsf{Fin}\; m \;\to\; \mathsf{Term}\; n \qquad g \;:\; \mathsf{Fin}\; m \;\to\; \mathsf{Term}\; n'}{f \le g \;:\; \mathsf{Type}}$$

$$f \le g \;\mapsto\; \exists f'.\, f \doteq f' \diamond g$$

$$\text{fact} \quad f \le f \qquad \frac{f \le g \qquad g \le h}{f \le h} \qquad f \le \iota \qquad \frac{f \le g}{f \diamond h \le g \diamond h}$$

We can now say what it means to be a *maximal* substitution satisfying P:

$$\text{let} \quad \frac{P \;:\; \mathbf{Property}\; m}{\mathbf{Max}\; P \;:\; \mathbf{Property}\; m} \qquad \mathbf{Max}\; P\; f \;\mapsto\; \begin{array}{l} P\; f\; \times \\ \forall f'.\, P\; f' \;\to\; f' \le f \end{array}$$

$$\text{fact} \quad \frac{P \Leftrightarrow Q}{\mathbf{Max}\; P \Leftrightarrow \mathbf{Max}\; Q}$$

Our unification algorithm finds maximal solutions by decomposing problems recursively and solving the subproblems sequentially, *accumulating* the solution. How do we know we end up with the maximal solution to the whole problem? In fact, this finds maximal substitutions for any property which is *downward-closed* in the following sense:

$$\text{let}\quad \frac{P \ : \ \mathbf{Property}\ m}{\mathbf{DClosed}\, P \ : \ \mathsf{Type}} \qquad \mathbf{DClosed}\, P \ \mapsto\ \forall f, g.\, f \leq g \ \rightarrow\ P\ g \ \rightarrow\ P\, f$$

$$\text{fact}\quad \frac{s, t \ : \ \mathsf{Term}\ m}{\mathbf{DClosed}\,(\mathbf{Unifies}\ s\ t)}$$

The following result shows that we can maximize a collection of downward-closed properties by accumulating a solution, exactly as **amgu** does. It says that if we want to extend some a until it just satisfies both P and Q, we may first extend it by p until it just satisfies P, then *further* by q until it just satisfies Q.

Lemma 1 (*The Optimist's Lemma*)

$$\text{fact}\quad \frac{\mathbf{DClosed}\, P \qquad \mathbf{Max}\ P[\_ \diamond a]\ p \qquad \mathbf{Max}\ Q[\_ \diamond (p \diamond a)]\ q}{\mathbf{Max}\ (P \wedge Q)[\_ \diamond a]\ (q \diamond p)}$$

Proof

Unpacking the premises, we know the following:

$$\begin{aligned} DCP &: \forall f, g.\, f \leq g \ \rightarrow\ P\ g \ \rightarrow\ P\, f \\ Ppa &: P\ (p \diamond a) \\ pMax &: \forall p'. P\ (p' \diamond a) \ \rightarrow\ p' \leq p \\ Qqpa &: Q\ (q \diamond p \diamond a) \\ qMax &: \forall q'. Q\ (q' \diamond p \diamond a) \ \rightarrow\ q' \leq q \end{aligned}$$

Unpacking the conclusion, we must show

$$\begin{array}{r} P\ (q \diamond p \diamond a) \\ Q\ (q \diamond p \diamond a) \\ \forall f. P\ (f \diamond a) \ \rightarrow\ Q\ (f \diamond a) \ \rightarrow\ f \leq q \diamond p \end{array}$$

$Q\,(q \diamond p \diamond a)$ is already known, whilst downward closure (DCP) reduces $P\,(q \diamond p \diamond a)$ to $P\ (p \diamond a)$.

For the third goal, suppose such an f exists.

By $pMax$, given $P\ (f \diamond a)$	$f \leq p$
So for some g	$f \doteq g \diamond p$
Hence (rewriting)	$Q\ (g \diamond p \diamond a)$
So by $qMax$	$g \leq q$
and for some h	$g \doteq h \diamond q$
That is,	$f \doteq h \diamond q \diamond p$
Hence	$f \leq q \diamond p$ $\square$

We must also check that failure of a subproblem leads to failure of the whole.

Lemma 2 (*The Failure Propagation Lemmas*)

$$\text{fact} \quad \frac{\mathbf{Nothing}\ P[\_ \diamond a]}{\mathbf{Nothing}\ (P \wedge Q)[\_ \diamond a]} \qquad \frac{\mathbf{Max}\ P[\_ \diamond a]\ p \quad \mathbf{Nothing}\ Q[\_ \diamond (p \diamond a)]}{\mathbf{Nothing}\ (P \wedge Q)[\_ \diamond a]}$$

Proof

The first is trivial. For the second, observe that any extension of a which solves P must actually extend $p \diamond a$ and thus fail to solve Q. $\square$

3 Simple Unification Problems

Sooner or later, we must actually generate some real substitutions to accumulate. The successful base cases of **amgu** are either trivial, or they eliminate one variable. It will thus be useful to know the following:

Lemma 3 (*The Trivial Problem Lemma*)

$$\text{fact} \quad \mathbf{Max}\ (\mathbf{Unifies}\ t\ t)[\_ \diamond f]\ \iota$$

Proof

Clearly, any substitution solves the problem—the identity is maximal. $\square$

Lemma 4 (*The Variable Elimination Lemma*)

$$\text{fact} \quad \mathbf{Max}\ (\mathbf{Unifies}\ (\iota\ x)\ (\rangle\, \mathbf{thin}\ x \,\langle t'))\ (t'\ \mathbf{for}\ x)$$

Proof

We have already seen that $(t'\ \mathbf{for}\ x)$ is one solution to this class of unification problem, but is it the most general? We must show that any other unifier f factors through it. In fact

$$f \doteq (f \cdot \mathbf{thin}\ x) \diamond t'\ \mathbf{for}\ x$$

This holds because each variable in the source set is either x itself or some $\mathbf{thin}\, x\, y'$. In the first case,

$$((f \cdot \mathbf{thin}\ x) \diamond t'\ \mathbf{for}\ x)\ x \;=\; (f \cdot \mathbf{thin}\ x) \langle\!\langle t' \;=\; f \langle\!\langle (\rangle\, \mathbf{thin}\ x \,\langle t') \;=\; f\ x$$

In the second, as $t'\ \mathbf{for}\ x \cdot \mathbf{thin}\ x \doteq \iota$

$$((f \cdot \mathbf{thin}\ x) \diamond t'\ \mathbf{for}\ x)\ (\mathbf{thin}\ x\ y') \;=\; (f \cdot \mathbf{thin}\ x) \langle\!\langle (\iota\ y') \;=\; f\ (\mathbf{thin}\ x\ y')$$

Hence $f \leq t'\ \mathbf{for}\ x$. $\square$

4 The Occur-Check

In order to explain why unification fails when **check** says no, we shall need to describe *occurrences*. In (McBride, 2001) I give a generic account of subterm occurrences for a given inductive datatype, by showing how to compute its associated type of the *one-hole contexts* where subtems can appear, based on Huet's notion of 'zipper' (Huet, 1997).

This datatype represents the path taken from the root of the term to the 'hole' where the subterm belongs, but also stores everything branching off that path on either side. Such a path in $\mathsf{Term}\ n$ is a sequence of *steps*, each taking us either to the left or the right subterm of a fork. As we take a step, we must record the other subterm. Hence steps are given by

$$\text{data} \quad \frac{n \ : \ \mathbb{N}}{\mathsf{Step}\ n \ : \ \mathsf{Type}} \quad \text{where} \quad \frac{r \ : \ \mathsf{Term}\ n}{\mathsf{left}\ r \ : \ \mathsf{Step}\ n} \quad \frac{l \ : \ \mathsf{Term}\ n}{\mathsf{right}\ l \ : \ \mathsf{Step}\ n}$$

A one-hole context in a term is thus given by $\mathsf{List}(\mathsf{Step}\,n)$. It is useful to see contexts as a *monoid* in the usual way: $[]$ is the 'identity' context, whilst composition is given by $+\!\!+$. This monoid has an *action* on terms—the action which plugs a term in the hole.

$$\text{let} \quad \frac{ps \ : \ \mathsf{List}\ (\mathsf{Step}\ n) \qquad t \ : \ \mathsf{Term}\ n}{ps + t \ : \ \mathsf{Term}\ n}$$

$$\begin{aligned} [] + t &\mapsto t \\ (\mathsf{left}\ r :: ps) + t &\mapsto (ps + t)\ \mathsf{fork}\ r \\ (\mathsf{right}\ l :: ps) + t &\mapsto l\ \mathsf{fork}\ (ps + t) \end{aligned}$$

It is easy to check that

$$\text{fact} \quad (ps +\!\!+ qs) + t \ = \ ps + (qs + t)$$

Furthermore, we can easily lift substitution to one-hole contexts $f \lessdot ps$ by applying f to each term contained by ps, and prove (by induction on ps) that

$$\text{fact} \quad f \lessdot (ps + t) \ = \ (f \lessdot ps) + (f \lessdot t)$$

Now we can characterize **check**:

$$\text{fact} \quad \frac{\textbf{check}\ x\ t \ = \ r}{\bigvee \begin{cases} \exists t'.\ \ t = \rhd \textbf{thin}\ x \lessdot t' \wedge r = \mathsf{yes}\ t' \\ \exists ps. \quad t = ps + \iota\ x \wedge r = \mathsf{no} \end{cases}}$$

That is, when **check** x t succeeds, it shows that $(\iota\ x)$ and t have a most general unifier by the Variable Elimination lemma.

On the other hand, if **check** x t fails, we are trying to find an f such that

$$f\ x \ = \ f \lessdot (ps + \iota\ x) \ = \ (f \lessdot ps) + f\ x$$

The following lemma shows that a solution exists only if ps is trivial.

Lemma 5 (*The No-Cycle Lemma*)

$$\text{fact} \quad \frac{t \;=\; ps + t}{ps \;=\; []}$$

Proof

Induction on t; case analysis on ps. The only hard cases require us to contradict

$$s \;\mathsf{fork}\; t \;=\; (\mathsf{left}\; r :: ps) + (s \;\mathsf{fork}\; t) \;\mapsto\; (ps + (s \;\mathsf{fork}\; t))\;\mathsf{fork}\; r$$

and similarly when the first step goes right. By injectivity of fork

$$s \;=\; ps + (s \;\mathsf{fork}\; t) \;=\; (ps +\!\!+ [\mathsf{left}\; t]) + s$$

The inductive hypothesis now yields

$$(ps +\!\!+ [\mathsf{left}\; t]) \;=\; []$$

which is clearly false. □

In particular, we now know that

$$\text{fact} \quad \mathbf{Nothing}\;(\mathbf{Unifies}\;(\iota\; x)\;((p :: ps) + \iota\; x))$$

5 Proving mgu Correct

We now have the language and the technology with which to characterize **mgu**.

Theorem 6 (*Correctness of* **mgu**)

$$\text{fact} \quad \frac{\mathbf{mgu}_m\; s\; t \;=\; r}{\bigvee \left\{ \begin{array}{c} \exists n, \sigma.\; \mathbf{Max}\;(\mathbf{Unifies}\; s\; t)\;(\mathbf{sub}\;\sigma) \wedge r = \mathsf{yes}\;\langle n, \sigma\rangle \\ \mathbf{Nothing}\;(\mathbf{Unifies}\; s\; t) \wedge r = \mathsf{no} \end{array}\right.}$$

This follows from the correctness of **amgu**:

Theorem 7 (*Correctness of* **amgu**)

$$\text{fact} \quad \frac{\mathbf{amgu}_m\; s\; t\; \langle l, \rho\rangle \;=\; r}{\bigvee \left\{ \begin{array}{c} \exists n, \sigma.\; \mathbf{Max}\;(\mathbf{Unifies}\; s\; t)[\_ \diamond \mathbf{sub}\;\rho]\;(\mathbf{sub}\;\sigma) \wedge r = \mathsf{yes}\;\langle n, \sigma +\!\!+ \rho\rangle \\ \mathbf{Nothing}\;(\mathbf{Unifies}\; s\; t)[\_ \diamond \mathbf{sub}\;\rho] \wedge r = \mathsf{no} \end{array}\right.}$$

Proof of Theorem 6

Suppose $\qquad \mathbf{mgu}_m\ s\ t\ =\ r$
By definition of **mgu**, $\quad \mathbf{amgu}_m\ s\ t\ \langle m, \mathsf{anil}\rangle\ =\ r$

We have initialized the accumulator with the identity substitution. Hence, by Theorem 7, there are two possibilities:

$$\bigvee \left\{ \begin{array}{r} \exists n, \sigma.\ \mathbf{Max}\ (\mathbf{Unifies}\ s\ t)[\_ \diamond \mathbf{sub}\, \mathsf{anil}]\ (\mathbf{sub}\, \sigma) \wedge r = \mathsf{yes}\ \langle n, \sigma +\!\!+ \mathsf{anil}\rangle \\ \mathbf{Nothing}\ (\mathbf{Unifies}\ s\ t)[\_ \diamond \mathbf{sub}\, \mathsf{anil}] \wedge r = \mathsf{no} \end{array} \right.$$

Recalling that $\mathbf{sub}\, \mathsf{anil} \mapsto \iota$, $\sigma +\!\!+ \mathsf{anil} \mapsto \sigma$ and $P[\_ \diamond \iota] \Leftrightarrow P$, we recover exactly

$$\bigvee \left\{ \begin{array}{r} \exists n, \sigma.\ \mathbf{Max}\ (\mathbf{Unifies}\ s\ t)\ (\mathbf{sub}\, \sigma) \wedge r = \mathsf{yes}\ \langle n, \sigma\rangle \\ \mathbf{Nothing}\ (\mathbf{Unifies}\ s\ t) \wedge r = \mathsf{no} \end{array} \right.$$

as required. □

Proof of Theorem 7

As we know **amgu** terminates, we may prove its correctness by induction on its *course of recursion*.

The successful base cases deliver substitutions which are most general unifiers either by the Trivial Problem lemma (for equal variables and leaf-leaf), or by the Variable Elimination lemma (for different variables and the **flexRigid** $x\ t$ cases when **check** $x\ t\ =\ \mathsf{yes}\ t'$).

The base cases which fail address rigid problems with unlike constructors, or the failure of the occur-check. It is easy to see that

$$\mathbf{Nothing}\ (\mathbf{Unifies}\ \mathsf{leaf}\ (t_1\ \mathsf{fork}\ t_2))$$

and its symmetric image. Meanwhile, **flexRigid** $x\ t$ is only called when t is not a variable, hence if **check** $x\ t = \mathsf{no}$, it must be the case that $t = (p :: ps) + (\iota\ x)$. We have already seen that this t does not unify with $(\iota\ x)$.

The step case which applies part of the accumulated substitution propagates both failure and success correctly from the inductive hypothesis for the recursive call because both **Nothing** and **Max** respect the following equivalence:

$$(\mathbf{Unifies}\ s\ t)[\_ \diamond \mathbf{sub}\ (\sigma\ \mathsf{asnoc}\ r/z)] \Leftrightarrow (\mathbf{Unifies}\ (r\ \mathbf{for}\ z \lhd\!\!\!\lhd s)\ (r\ \mathbf{for}\ z \lhd\!\!\!\lhd t))[\_ \diamond \mathbf{sub}\, \sigma]$$

The step case for $\mathbf{amgu}_m\ (s_1\ \mathsf{fork}\ s_2)\ (t_1\ \mathsf{fork}\ t_2)\ \langle n, \sigma\rangle$ splits this unification problem into an equivalent pair of subproblems:

$$(\mathbf{Unifies}\ (s_1\ \mathsf{fork}\ s_2)\ (t_1\ \mathsf{fork}\ t_2)) \Leftrightarrow (\mathbf{Unifies}\ s_1\ t_1) \wedge (\mathbf{Unifies}\ s_2\ t_2)$$

These are then solved in the optimistic style. The corresponding inductive hypotheses tell us either that one of these subproblems has no solution and that no is returned, or else a substitution is successfully accumulated which is most general for the subproblems in turn. In the former case, the Failure Propagation lemmas tell us that no is the correct answer for the whole problem. In the latter, the accumulated substitution is most general by the Optimist's lemma. □

References

Huet, Gérard. (1997). The Zipper. *Journal of Functional Programming*, **7**(5), 549–554.

McBride, Conor. (2001). *The Derivative of a Regular Type is its Type of One-Hole Contexts.* Electronically available: `http://www.dur.ac.uk/c.t.mcbride/diff.ps`.