A Machine Checked Model of Idempotent MGU Axioms For a List of Equational Constraints

Sunil Kothari, James Caldwell
Department of Computer Science,
University of Wyoming, USA

Machine checked proofs of type inference algorithms often axiomatize MGU behavior as a set of axioms. Idempotent MGUs for a list of equational constraints are needed to reason about the correctness of Wand’s type inference algorithm and our extension of it. To characterize the behavior of idempotent MGUs, we propose a set of seven axioms; four of which have been proven in our earlier paper, where we formally verify that the first order unification is a model for the axioms. This paper shows that the first order unification is a model for the remaining three idempotent MGU axioms. Coq’s method of functional induction is the main proof technique used in proving the axioms.

1 Introduction

Even though there are formalizations of the unification algorithm in a number of different theorem provers, many of the existing verifications of type inference algorithms [3, 8, 10] axiomatize the behavior of a most general unifier (MGU) as a set of axioms. The MGU axioms for a list of constraints are needed for constraint-based type inference algorithms, for instance, Wand’s type inference algorithm [11]. Here we use a modified version of the MGU axioms given in [10]. The axioms there are stated for a single constraint, whereas we state axioms for a list of constraints. Let $\rho, \rho'$ denote substitutions i.e. functions mapping type variables to terms and $C$ denotes a constraint list, then the MGU axioms (for a list of constraints) are:

(i) $\text{mgu} \; \rho \; C \Rightarrow \rho \models C$
(ii) $\text{mgu} \; \rho \; C \land \rho' \models C \Rightarrow \exists \rho''. \; \rho' \approx \rho \circ \rho''$
(iii) $\text{mgu} \; \rho \; C \Rightarrow \text{FTV} (\rho) \subseteq \text{FTV} (C)$
(iv) $\rho \models C \Rightarrow \exists \rho'. \; \text{mgu} \; \rho' \; C$

A little note on the terminologies and the notations used in the paper. FTV is used to denote the free type variables of a type, a substitution, and a constraint list. $\models$ denotes constraint satisfiability. $\approx$ denotes extensional equality on substitutions. List append is denoted by $++$ and the empty substitution is denoted by $Id$. Furthermore, we have adopted the following conventions in this paper: atomic types are denoted by $\alpha, \beta, \alpha'$ etc.; compound types by $\tau, \tau', \tau_1$ etc.; substitutions by $\rho, \rho', \rho_1$ etc.; and finite maps by $\sigma, \sigma', \sigma_1$ etc. Small finite maps will be represented using the usual (enumerative) set notation. For example, a finite map that binds $\alpha$ to $\tau$ is denoted as $\{ \alpha \mapsto \tau \}$. $\emptyset$ denotes the empty finite map.

Although the above set of axioms can model MGU for substitution-based type inference algorithms, our experience has been that the constraint-based type inference algorithms require reasoning about a restricted set of MGUs, namely the idempotent MGUs. The idempotent MGUs have the nice property that their domain and range elements are disjoint. This property is useful in our proofs of correctness of Wand’s type inference algorithm. To the four MGU axioms mentioned above, we add three more axioms to characterize idempotent MGUs for a list of equational constraints:
We have verified the following in Coq:

\[
\begin{align*}
  (v) & \quad \text{mgu } \rho \ C \Rightarrow \rho \circ \rho \approx \rho \\
  (vi) & \quad \text{mgu } \rho \ [] \Rightarrow \rho = \text{Id} \\
  (vii) & \quad \text{mgu } \rho' \ C' \land \text{mgu } \rho'' (\rho'(C'')) \land \text{mgu } \rho (C' + + C'') \Rightarrow \rho \approx \rho' \circ \rho''
\end{align*}
\]

Previously, we showed that the first-order unification algorithm (with finite maps representing substitutions) is a model for MGU axioms [6]. In this paper we extend the axioms set to characterize idempotent MGUs, and then we show that the first-order unification remains a model for the extended set. We have verified the following in Coq:

\[
\begin{align*}
  (i) & \quad \text{unify } C = \sigma \Rightarrow \sigma \vdash C \\
  (ii) & \quad (\text{unify } C = \sigma \land \sigma' \vdash C) \Rightarrow \exists \sigma''. \; \sigma' \approx \sigma \circ \sigma'' \\
  (iii) & \quad \text{unify } C = \sigma \Rightarrow \text{FTV}(\sigma) \subseteq \text{FTV}(C) \\
  (iv) & \quad C \vdash \exists \sigma'. \; \text{unify } C = \sigma' \\
  (v) & \quad \text{unify } C = \sigma \Rightarrow \sigma \circ \sigma \approx \sigma \\
  (vi) & \quad \text{unify } [] = \sigma \Rightarrow \sigma = \sigma_{\text{def}} \\
  (vii) & \quad (\text{unify } C' = \sigma' \land \text{unify } \sigma'(C'') = \sigma'' \land \text{unify } (C' + + C'') = \sigma) \Rightarrow \sigma \approx \sigma' \circ \sigma''
\end{align*}
\]

In this paper, we provide a proof sketch for Axiom v, vi and vii. Also, note that unify denotes the first order unification algorithm, which is mentioned later in Section 2.

This work is in the context of a formal verification in Coq [9] of our extension of Wand’s constraint-based type inference algorithm to the polymorphic let construct [5]. Coq is a constructive theorem prover based on the calculus of inductive constructions [2]. While formalizing, we realized that Urban and Nipkow’s axioms do not characterize idempotent MGUs on a list of constraints. In particular, we wanted an axiom which could relate constraint satisfiability of a list to its sub lists. The theorems stated here are also mentioned in the unification literature, namely [4,7], but without proofs.

## 2 Substitutions, Unification and Functional Induction

Unification is implemented here over a language of types for (untyped) lambda terms. The language of types is given by the following grammar:

\[
\tau ::= \alpha \mid \tau_1 \to \tau_2
\]

where \(\alpha\) is a type variable and \(\tau_1, \tau_2 \in \tau\) are type terms.

Thus, a type is either a type variable or a function type. We believe that our proofs can be easily extended to the generalized term structure. The constraints are of the form \(\tau \equiv \tau'\). And substitutions are represented as finite functions (a.k.a finite maps in Coq) - mapping type variables to types. We use Coq’s finite map library \texttt{Coq.FSets.FMapInterface}, which provides an axiomatic presentation of finite maps and a number of supporting implementations.

Application of a finite map to a type, and to a constraint is defined as:

\[
\begin{align*}
  \sigma(\alpha) & \overset{\text{def}}{=} \text{if } \langle \alpha, \tau \rangle \in \sigma \text{ then } \tau \text{ else } \alpha \\
  \sigma(\tau_1 \to \tau_2) & \overset{\text{def}}{=} \sigma(\tau_1) \to \sigma(\tau_2) \\
  \sigma(\tau_1 \equiv \tau_2) & \overset{\text{def}}{=} \sigma(\tau_1) \equiv \sigma(\tau_2)
\end{align*}
\]

The equality on finite maps is dependent on their representation - we use an extensional equality to abstract from the actual representation.

\[
\sigma \approx \sigma' \overset{\text{def}}{=} \forall \alpha. \; \sigma(\alpha) = \sigma'(\alpha)
\]

Moreover, the equality can be extended to all types as given by the following lemma:

**Lemma 1.** \(\forall \alpha. \; \sigma(\alpha) = \sigma'(\alpha) \iff \forall \tau. \; \sigma(\tau) = \sigma'(\tau)\)

From our definition of composition of finite maps (see [6] for definition), we prove the following lemma:
Lemma 2. ∀σ, σ'. ∀τ. (σ ∘ σ')(τ) = σ'(σ(τ))

We use the following standard presentation of the first-order unification algorithm.

\[
\begin{align*}
\text{unify } (\alpha \equiv \alpha) & : \equiv \text{unify } \alpha \\
\text{unify } (\alpha \equiv \beta) & : \equiv \{\alpha \mapsto \beta\} \circ \text{unify } (\{\alpha \mapsto \beta\} \circ \alpha) \\
\text{unify } (\alpha \equiv \tau) & : \equiv \begin{cases} \\
\text{if } \alpha \text{ occurs in } \tau \text{ then Fail else } \{\alpha \mapsto \tau\} \circ \text{unify } (\{\alpha \mapsto \tau\} \circ \alpha) \\
\end{cases} \\
\text{unify } (\tau \equiv \alpha) & : \equiv \begin{cases} \\
\text{if } \alpha \text{ occurs in } \tau \text{ then Fail else } \{\alpha \mapsto \tau\} \circ \text{unify } (\{\alpha \mapsto \tau\} \circ \alpha) \\
\end{cases} \\
\text{unify } ([\ ]) & : \equiv \sigma_\bot \\
\end{align*}
\]

Our Coq specification of the first order unification (see Appendix in [6]) models the failure as an option type - with failure represented as None, and success represented as Some(σ), where σ is the resulting substitution. In the proofs, we show only the important cases and the failure cases are not shown.

The underlying theme in all of the proofs below is the use of functional induction technique [1] in Coq. This induction is stronger than the normal list induction, and the induction principle closely follows the actual syntax of the definition. In Coq, the functional induction tactic is defined as a wrapper for repeated inductions.

functional induction \( (f \ x_1 \ x_2 \ldots x_n) \equiv \text{induction } x_1 \ x_2 \ldots x_n(f \ x_1 \ x_2 \ldots x_n) \) using \( f \_\text{ind} \)
where \( f \_\text{ind} \) is the induction principle generated by Coq for \( f \). The technique is justified by the fact that Coq generates a set of proof obligations, which allows us to prove the termination separately. The actual induction principle, generated proof obligations, and the termination measure are available in our previous paper [6].

3 Proof Sketch of Axioms

In the next few sections, we give a proof sketch of the main theorem and also state the important lemmas.

3.1 Axiom v

The following lemmas are needed for the main proof.

Lemma 3. ∀σ. ∀α. ∀τ. α ∉ FT(p(τ)) ∧ α ∉ FT(p(σ)) ⇒ α ∉ FT(p(σ(τ)))

Lemma 4. ∀σ. ∀τ, τ'. α ∉ FT(p(τ)) ⇒ \{α \mapsto τ'\}(τ) = τ

Lemma 5. ∀σ. ∀τ, τ'. α ∉ FT(p(τ)) ⇒ α ∉ FT(\{α \mapsto τ\}(σ))

We must prove the following theorem:

Theorem 1. ∀C. ∀σ. unify C = σ ⇒ ∀σ. (σ ∘ σ)(σ) = σ(σ).

Proof. Choose an arbitrary C. By Lemma 3 we must show:

∀σ. unify C = σ ⇒ ∀σ. σ(σ(σ)) = σ(σ). By functional induction on unify C, we have two main cases:

Case C = [\ ]. This case follows since ∀σ. σ(σ(σ)) = σ(σ).

Case C ≠ [\ ]. We will have the following sub cases depending on the head of the constraint list:

1. Case (α ≡ α) :: C. Apply the induction hypothesis and then this case is trivial.
2. Case $(\alpha \equiv \beta) :: C$. Similar to the case below.
3. Case $(\alpha \equiv \tau_1 \rightarrow \tau_2) :: C$ and $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$. We know $\text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C)) = \sigma'$ and the induction hypothesis reads as:

$$\forall \sigma'. \text{unify} \{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C) = \sigma' \Rightarrow \forall \alpha'. \sigma' = \sigma'(\alpha')$$

And we must show

$$\sigma(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\alpha'')) = (\sigma(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\sigma(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\alpha'')))).$$

There are two cases:

(a) Case $\alpha = \alpha''$. Then we must show $\sigma(\tau_1 \rightarrow \tau_2) = \sigma(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\sigma(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\})))$. From Lemma 5 and Axiom iii, we know that $\alpha \not\in \text{FTV}(\sigma)$. Since $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$ and $\alpha \not\in \text{FTV}(\sigma')$, so by Lemma 3 $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$. By Lemma 4 and using $\tau'$ to be $\tau_1 \rightarrow \tau_2$ we get $\sigma'(\tau_1 \rightarrow \tau_2) = (\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\sigma'(\tau_1 \rightarrow \tau_2)))$. So now we must show $\sigma(\tau_1 \rightarrow \tau_2) = \sigma(\sigma(\tau_1 \rightarrow \tau_2))$. Then, by Lemma 1 we must show $\forall \beta. \sigma(\beta) = \sigma(\sigma(\beta))$.

(b) Case $\alpha \neq \alpha''$. Similar to the above case.
4. Case $(\tau_1 \rightarrow \tau_2 \equiv \alpha) :: C$ and $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$. Same as above.
5. Case $(\tau_1 \rightarrow \tau_2 \equiv \tau_3 \rightarrow \tau_4) :: C$. Apply the induction hypothesis and then this case is trivial.

3.2 Axiom vi

The theorem we must prove is:

**Theorem 2.** $\forall \sigma. \text{unify}[] = \sigma \Rightarrow \sigma = \sigma_\Xi$

**Proof.** Choose an arbitrary $\sigma$. Assume $\text{unify}[] = \sigma$. Unfold the definition of $\text{unify}$. Then we have $\sigma = \sigma_\Xi$. □

3.3 Axiom vii

The axiom proof requires lemmas as given below:

**Lemma 6.** $\forall \eta. \forall \eta'. \eta :: (C ++ C') = (\eta :: C) ++ C'$

**Lemma 7.** $\forall C. \forall \sigma. \forall x. \forall \tau. \sigma(\{x \mapsto \tau\}(C)) = (\{x \mapsto \tau\} \circ \sigma)(C)$

**Lemma 8.** $\forall C, C'. \forall x. \forall \tau. \{x \mapsto \tau\}(C) ++ \{x \mapsto \tau\}(C') = \{x \mapsto \tau\}(C ++ C')$

The theorem we must prove is:

**Theorem 3.** $\forall C, C'. \forall \sigma, \sigma', \sigma''. (\text{unify } C = \sigma' \land \text{unify } \sigma'(C') = \sigma'' \land \text{unify } (C ++ C') = \sigma) \\
\Rightarrow \forall \beta'. \sigma(\beta') = (\sigma' \circ \sigma'')(\beta')$

**Proof.** Choose an arbitrary $C$. By Lemma 2 we must show:

$$\forall C'. \forall \sigma, \sigma', \sigma''. (\text{unify } C = \sigma' \land \text{unify } \sigma'(C') = \sigma'' \land \text{unify } (C ++ C') = \sigma) \\
\Rightarrow \forall \beta'. \sigma(\beta') = (\sigma''(\sigma'(\beta')))$$

By functional induction on $\text{unify } C$, we have two main cases:

Case $C = [\ ]$. Follows from Theorem 2 and the assumptions.
Case $C \neq [\ ]$. We will have the following cases based on the constraint at the head of the constraint list.
1. Case ($\alpha \equiv \alpha$) :: C. Use the induction hypothesis and Lemma 6.

2. Case ($\alpha \equiv \beta$) :: C and $\alpha \not\equiv \beta$. Similar to the case below.

3. Case ($\alpha \equiv \tau_1 \rightarrow \tau_2$) :: C and $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$. Assume $\text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C)) = \sigma'$, $\text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\} \circ \sigma')(C') = \sigma''$ and $\text{unify}(\{(\alpha \equiv \tau_1 \rightarrow \tau_2) :: C \mathbin{\leftarrow \mathbin{\leftarrow}{\mathbin{\leftarrow} C}'\rangle}) = \sigma$. By Lemma 6, the last assumption is $\text{unify}(\{(\alpha \equiv \tau_1 \rightarrow \tau_2) :: C \mathbin{\leftarrow \mathbin{\leftarrow}{\mathbin{\leftarrow} C}'\rangle}) = \sigma$. Unfolding the unify definition once, assume $\text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C \mathbin{\leftarrow \mathbin{\leftarrow}{\mathbin{\leftarrow} C}'\rangle) = \sigma_T$. Then $\sigma = \{\alpha \mapsto \tau_1 \rightarrow \tau_2\} \circ \sigma_T$. We must show $\{\alpha \mapsto \tau_1 \rightarrow \tau_2\} \circ \sigma_T(\beta') = \sigma''(\sigma' \{\alpha \mapsto \tau_1 \rightarrow \tau_2\} (\beta'))$. By Lemma 2, $\sigma_T(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(\beta')) = \sigma''(\sigma' (\{\alpha \mapsto \tau_1 \rightarrow \tau_2\} (\beta'))).$ We have two cases:
   (a) Case $\alpha = \beta'$. The induction hypothesis is:
   $\forall C_1. \forall \sigma_1, \sigma_2, \sigma_3. \text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C_1)) = \sigma_1 \land \text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C_1 \mathbin{\leftarrow \mathbin{\leftarrow}{\mathbin{\leftarrow} C}'\rangle) = \sigma_2 \land \text{unify}(\{\alpha \mapsto \tau_1 \rightarrow \tau_2\}(C \mathbin{\leftarrow \mathbin{\leftarrow}{\mathbin{\leftarrow} C}'\rangle) = \sigma_3 \Rightarrow \forall \beta''. \sigma_1(\beta'') = \sigma_2(\sigma_T(\beta'))$.
   Then we must show $\sigma_T(\tau_1 \rightarrow \tau_2) = \sigma''(\sigma' (\tau_1 \rightarrow \tau_2))$. But by Lemma 1, $\forall \gamma. \sigma_T(\gamma) = \sigma''(\sigma'(\gamma))$. Choose an arbitrary $\gamma$ and so we must show $\sigma_T(\gamma) = \sigma''(\sigma'(\gamma))$. But that follows from the induction hypothesis and Lemma 7 and Lemma 8 and the assumptions.
   (b) Case $\alpha \not\equiv \beta'$. Similar to the case above.

4. Case ($\tau_1 \rightarrow \tau_2 \equiv \alpha$) :: C and $\alpha \not\in \text{FTV}(\tau_1 \rightarrow \tau_2)$. Same as the above case.

5. Case ($\tau_1 \rightarrow \tau_2 \equiv \tau_3 \rightarrow \tau_4$) :: C. Apply the induction hypothesis.

The entire formalization (all seven axioms) is done in Coq 8.1.pl3 version in around 5000 lines of specifications and tactics, and is available online at [http://www.cs.uwyo.edu/~skothari](http://www.cs.uwyo.edu/~skothari). Lastly, we would like to thank Christian Urban (TU Munich) for discussing at length about the MGU axioms used in their verification of Algorithm W.

References


