Unification for the Functional Machine Calculus

June 28, 2023

Introduction

todo

Unification

The *unification calculus* is given as follows. For a first-order signature Σ consisting of a set Σ_n of terms for each *arity* $n \in \mathbb{N}$, *values* are first-order terms t, u, as given by the first grammar below, and *computation terms* M, N are given by the second grammar.

$$\begin{array}{lll} t,\,u & ::= & x \mid f(t_1,\dots,t_n)\ (f \in \Sigma_n) \\ M,\,N & ::= & \star \mid \langle t\rangle.\,M \mid [t].\,M \mid \nu x.\,M \end{array}$$

The *unification machine* is given by the following transition rules:

$$\frac{(\ S\ ,\ [t].\,M\ )}{(\ S\,t\ ,\ M\ )} \tag{1}$$

$$\frac{(\ S\,f(t_1,\dots,t_n)\ ,\ \langle f(u_1,\dots,u_n)\rangle.\,M\ )}{(\ S\,t_n\dots t_1\ ,\ \langle u_1\rangle\dots\langle u_n\rangle.\,M\ )} \tag{2}$$

$$\frac{(\ S\,x\ ,\ \langle x\rangle.\,M\ )}{(\ S\,t\ ,\ M\ )} \tag{3}$$

$$\frac{(\ S\,t\ ,\ \langle x\rangle.\,M\ )}{(\ \{t/x\}S\ ,\ \{t/x\}M\ )} \qquad \frac{(\ S\,x\ ,\ \langle t\rangle.\,M\ )}{(\ \{t/x\}S\ ,\ \{t/x\}M\ )} \qquad (x \notin t) \tag{4}$$

$$\frac{(\ S\ ,\ \nu x.\,M\ )}{(\ S\ ,\ \{y/x\}M\ )} \qquad (y \notin S, M) \tag{5}$$

Proposition 1. *For t and u with free variables x_1, ..., x_n,*

$$\frac{(\ x_1\dots x_n\,t\ ,\ \langle u\rangle\ )}{(\ w_1\dots w_n\ ,\ \star\ )}$$

if and only if $\{w_i/x_i\}_{i\le n}$ is a most general unifier of t and u.

Proof. Each state in the run of $(x_1 \dots x_n\, t, \langle u\rangle)$ is of the form $(v_1 \dots v_n\, t_m \dots t_1, \langle u_1\rangle \dots \langle u_m\rangle)$. We first define a map U sending a state to a stack of equations $E = [t_1 = u_1, \dots, t_m = u_m]$ and substitution $\theta = \{\{v_1/x_1\}, \dots, \{v_n/x_n\}\}$. Note that rules (2), (3) and (4) then correspond to those of the unification algorithm given in [Sterling & Shapiro, 1986] (modulo occurrences of $\{x/x\}$ in the substitution) under this translation. Given two first-order terms t and u, the algorithm is initialized with the stack $E = [t = u]$ and substitution $\theta = \{\}$. Termination with `failure=false` then occurs iff E reaches the empty stack, in which case $\theta = \{\{w_1/x_1\}, \dots, \{w_n/x_n\}\}$ is the most general unifier of t and u. Since the translation of $(x_1 \dots x_n\, t, \langle u\rangle)$ corresponds to the initialization of this algorithm, the result follows by induction and noting that if the algorithm terminates successfully on (E, θ), then $U^{-1}((E,\theta)) = (w_1 \dots w_n, \star)$.

□

Interaction nets

The interaction calculus [Fernandez & Mackie, 1999] describes interaction nets as follows. Given a first-order signature Σ, a *net* is a pair $\langle \vec{t} \mid \Delta \rangle$ where $\vec{t}$ is a vector of first-order terms over Σ, and $\Delta = \{t_1 = u_1, \dots, t_n = u_n\}$ is a set of equations between terms such that each variable occurs at most twice in the net. For vectors $\vec{t}$ and $\vec{u}$ of equal length, we may write $\vec{t} = \vec{u}$ for their pairwise equations.

Nets are rewritten according to a set of rules $\mathcal{R}$, given by pairs of terms $f(\vec{s}) \bowtie g(\vec{u})$ where $f \neq g$ and where each occuring variable occurs exactly twice, and such that there is at most one rule for each combination f, g in $\mathcal{R}$. Rules are considered modulo renaming of variables, and to use fresh variables when they are applied.

The evaluation relation $\twoheadrightarrow_{\mathcal{R}}$ for the rule set $\mathcal{R}$ is generated by the following two rules, adapted from the four of [Fernandez & Mackie, 1999].

$$\begin{aligned} \langle \vec{t} \mid \Delta, x = u \rangle &\twoheadrightarrow \langle \{u/x\}\vec{t} \mid \{u/x\}\Delta \rangle \\ \langle \vec{t} \mid \Delta, f(\vec{r}) = g(\vec{v}) \rangle &\twoheadrightarrow \langle \vec{t} \mid \Delta, \vec{r} = \vec{s}, \vec{u} = \vec{v} \rangle \quad (f(\vec{s}) \bowtie g(\vec{u}) \in \mathcal{R}) \end{aligned}$$

The interaction calculus embeds into the unification calculus by taking an equation $t = u$ to a redex $[t].\langle u\rangle$. Given a set of ordered equations Δ, the map ψ is defined as follows:

$$t_1 = u_1, \dots, t_n = u_n \mapsto [t_1].\langle u_1\rangle \dots [t_n].\langle u_n\rangle$$

The translation then sends $\langle \vec{s} \mid \Delta \rangle$ to $(\vec{s}, \psi(\Delta))$. To evaluate a translated net on the machine, for every rule $f(\vec{s}) \bowtie g(\vec{u}) \in \mathcal{R}$ with variables $\vec{x}$ we add the following machine step:

$$\frac{(\ S\, f(\vec{r})\ ,\ \qquad\qquad\qquad\qquad\qquad \langle g(\vec{v})\rangle.\, M\ )}{(\ S\, f(\vec{r})\ ,\ (\nu\vec{x}.\langle f(\vec{s})\rangle.[g(\vec{u})])\,;\langle g(\vec{v})\rangle.\, M\ )} \tag{6}$$

where the *new* construct $\nu x.\, M$ is used to capture the idea that variables in a rule $f(\vec{s}) \bowtie g(\vec{u})$ are bound. The machine evaluation relation $\Downarrow_{\mathcal{R}}$ consists of the basic rules for the unification machine plus the rules induced by $\mathcal{R}$. (Note that since rules are symmetric, for each rule $f(\vec{s}) \bowtie g(\vec{u})$ we also add the transition rule for $g(\vec{u}) \bowtie f(\vec{s})$.)

Note that

$$\frac{(\ S\, f(\vec{r})\ ,\ (\nu\vec{x}.\, \langle f(\vec{s})\rangle.\, [g(\vec{u})])\, ;\, \langle g(\vec{v})\rangle.\, M\ )}{(\ S\, f(\vec{r})\ ,\ \langle f(\vec{s})\rangle.\, [g(\vec{u})].\, \langle g(\vec{v})\rangle.\, M\ )}$$

where the free variables $\vec{x}$ in $f(\vec{s})$ and $g(\vec{u})$ are fresh. The rule (6) is therefore equivalent to

$$\frac{(\ S\, f(\vec{r})\ ,\ \langle g(\vec{v})\rangle.\, M\ )}{(\ S\, f(\vec{r})\ ,\ \langle f(\vec{s})\rangle.\, [g(\vec{u})].\, \langle g(\vec{v})\rangle.\, M\ )} \tag{7}$$

To define a read-back function, consider the following rewrite rule on tuples $(S \mid \Delta)$ of machine states and net equations:

$$\left.\begin{array}{l} ((\vec{t}\, a, \langle b\rangle.\, M) \mid \Delta) \\ ((\vec{t}, [a].\, \langle b\rangle.\, M) \mid \Delta) \end{array}\right\} \rightsquigarrow \begin{cases} ((\vec{t}, M) \mid \Delta, \vec{r} = \vec{v}) & \text{if } a = f(\vec{r}) \text{ and } b = f(\vec{v}) \\ ((\vec{t}, M) \mid \Delta, a = b) & \text{otherwise} \end{cases}$$

Since this rule is deterministic, it induces a partial function ϕ from the set of machine states to the set of nets, where $\phi((\vec{t}, M)) = \langle \vec{t} \mid \Delta\rangle$ whenever $((\vec{t}, M) \mid \varnothing) \rightsquigarrow^* ((, \star) \mid \Delta)$. Note in particular that $\phi((\vec{t}, \psi(\Delta))) = \langle \vec{t} \mid \Delta\rangle$.

Proposition 2. *For a set of rules $\mathcal{R}$ and a net $\langle \vec{t} \mid \Delta\rangle$, we have the following:*

$$(\vec{t}, \psi(\Delta)) \Downarrow_{\mathcal{R}} (\vec{u}, \star) \iff \langle \vec{t} \mid \Delta\rangle \twoheadrightarrow_{\mathcal{R}} \langle \vec{u} \mid\rangle$$

Forward direction: Our goal is to prove $(\vec{t}, \psi(\Delta)) \Downarrow_{\mathcal{R}} (\vec{u}, \star) \implies \langle \vec{t} \mid \Delta\rangle \twoheadrightarrow_{\mathcal{R}} \langle \vec{u} \mid\rangle$. We first show that for any machine state S such that $\phi(S)$ is defined, if $S \Downarrow_{\mathcal{R}} S'$ after some sequence of machine steps m then there exists a sequence of interaction net evaluations n acting on $\phi(S)$ such that the following diagram commutes:

$$\begin{tikzcd} S \arrow[r, "m"] \arrow[d, "\phi"'] & S' \arrow[d, "\phi"] \\ N \arrow[r, "n"] & N' \end{tikzcd} \tag{8}$$

Commutativity follows by case analysis on S:

Rule (1): If $S = (\vec{t}, [a].\, M)$, then $\phi(S) = \phi((\vec{t}\, a, M))$.

Rule (2): If $S = (\vec{t}\, f(\vec{r}), \langle f(\vec{v})\rangle.\, M)$, then $\phi(S) = \langle \vec{t} \mid \Delta, \vec{r} = \vec{v}\rangle$, and $\phi((\vec{t}\, r_1 \dots r_n, \langle v_1\rangle \dots \langle v_n\rangle.\, M)) = \langle \vec{t} \mid \Delta, \vec{r} = \vec{v}\rangle$.

Rules (3) and (4): If $S = (\vec{t}\, a, \langle x\rangle.\, M)$, then $\phi(S) = \langle \vec{t} \mid \Delta, a = x\rangle$, and we have $\phi((\{a/x\}\vec{t}, \{a/x\}M)) = \langle \{a/x\}\vec{t} \mid \{a/x\}\Delta\rangle$ and $\langle \vec{t} \mid \Delta, a = x\rangle \rightarrow \langle \{a/x\}\vec{t} \mid \{a/x\}\Delta\rangle$. Likewise for the cases $S = (\vec{t}\, x, \langle a\rangle.\, M)$ and $S = (\vec{t}\, x, \langle x\rangle.\, M)$.

Rule (5): If $S = (\vec{t}, \nu x.M)$, then $\phi(S)$ is not defined. Contradiction.

Rule (7): If $S = (\vec{t}\, f(\vec{r}), \langle g(\vec{v})\rangle.\, M)$, then we have $\phi(S) = \langle \vec{t} \mid \Delta, f(\vec{r}) = g(\vec{v})\rangle$ and that $f(\vec{s}) \bowtie g(\vec{u})$. So $\phi((\vec{t}\, f(\vec{r}), \langle f(\vec{s})\rangle.\, [g(\vec{u})].\, \langle g(\vec{v})\rangle.\, M)) = \langle \vec{t} \mid \Delta, \vec{r} = \vec{s}, \vec{u} = \vec{v}\}\rangle$ and $\langle \vec{t} \mid \Delta, f(\vec{r}) = g(\vec{v})\rangle \rightarrow \langle \vec{t} \mid \Delta, \vec{r} = \vec{s}, \vec{u} = \vec{v}\}\rangle$.

The result then follows by diagram pasting:

$$
\begin{tikzcd}
(\vec{t}, \psi(\Delta)) \arrow[r, "m_0"] \arrow[d, "\phi"] & S_1 \arrow[r, "m_1"] \arrow[d, "\phi"] & \dots \arrow[r, "m_k"] \arrow[d, "\phi"] & (\vec{u}, \star) \arrow[d, "\phi"] \\
\langle \vec{t} \mid \Delta\rangle \arrow[r, "n_0"] & N_1 \arrow[r, "n_1"] & \dots \arrow[r, "n_k"] & \langle \vec{u} \mid\rangle
\end{tikzcd}
$$

Backwards direction: We look to prove $\langle \vec{t} \mid \Delta\rangle \twoheadrightarrow_{\mathcal{R}} \langle \vec{u} \mid\rangle \implies (\vec{t}, \psi(\Delta)) \Downarrow_{\mathcal{R}} (\vec{u}, \star)$. Given an interaction net N, suppose $\phi^{-1}(N)$ is nonempty. We now show that either $N = \langle \vec{u} \mid\rangle$, or for all elements $S = c(\phi^{-1}(N))$ of $\phi^{-1}(N)$ there exists an N' such that $N \rightarrow N'$, a sequence of machine steps m acting on S, and an element $S' = c'(\phi^{-1}(N'))$ of $\phi^{-1}(N')$ such that the following diagram commutes:

$$
\begin{tikzcd}
N \arrow[r] \arrow[d, "c \circ \phi^{-1}"'] & N' \arrow[d, "c' \circ \phi^{-1}"] \\
S \arrow[r, "m"] & S'
\end{tikzcd}
$$

Suppose $N = \langle \vec{t} \mid \Delta\rangle$ with $\Delta \neq \varnothing$. Pick an element S of $\phi^{-1}(N)$ - if $S = (\vec{t}\, f(\vec{r}), \langle f(\vec{v})\rangle.\, M)$, apply transition rule (2) to get that $S \Downarrow_{\mathcal{R}} S'$ with $S' \in \phi^{-1}(N)$. By induction on tree heights, we are guaranteed to reach some $S'' \in \phi^{-1}(N)$ matching either case (4) or (7) in a finite number of machine steps. At this point, we may apply a final machine step to obtain the state S''' after some sequence of steps m acting on N, with $N' = \phi(S''')$ and $S''' = c'(\phi^{-1}(N'))$ making the diagram commute (by commutativity of (8).)

The evaluation relation $\rightarrow_{\mathcal{R}}$ is known to be confluent, so by strong normalisation of $\langle \vec{t} \mid \Delta\rangle$ we have that $\langle \vec{t} \mid \Delta\rangle$ will always reach $\langle \vec{u} \mid\rangle$ after finitely many applications of $\rightarrow$. Note also that

$(\vec{t}, \psi(\Delta)) \in \phi^{-1}(\langle \vec{t} \mid \Delta \rangle)$ and $\phi^{-1}(\langle \vec{u} \mid \rangle) = \{(\vec{u}, \star)\}$, so by setting $c_0 \circ \phi^{-1}(\langle \vec{t} \mid \Delta \rangle) = (\vec{t}, \psi(\Delta))$ and diagram pasting we obtain the result:

$$\begin{tikzcd}
\langle \vec{t} \mid \Delta \rangle \arrow[r] \arrow[d, "c_0 \circ \phi^{-1}"] & N_1 \arrow[r] \arrow[d, "c_1 \circ \phi^{-1}"] & \dots \arrow[r] \arrow[d, "c_{k-1} \circ \phi^{-1}"] & \langle \vec{u} \mid \rangle \arrow[d, "c_k \circ \phi^{-1}"] \\
(\vec{t}, {\color{red}\psi(\Delta)}) \arrow[r, "m_0"] & S_1 \arrow[r, "m_1"] & \dots \arrow[r, "m_{k-1}"] & (\vec{u}, {\color{red}\star})
\end{tikzcd}$$

□

Prolog

A model of Prolog can be defined using the following objects:

Terms:	$t ::= x \mid f(t_1, \dots, t_n)$
Atoms:	$A ::= P(t_1, \dots, t_n)$
Clauses:	$C ::= A :\!- A_1 \dots A_n$
Programs:	$L ::= C_1 \dots C_n$

The corresponding abstract interpreter is then given in [Sterling & Shapiro, 1986]:

Input: A query ?– A. and program L
Output: An instance of A that is a logical consequence of L, otherwise *fail*

$R \leftarrow \{G\}$
while $R \neq \varnothing$ **do**
  c hoose a goal B' from R
  if there exists a clause $B :\!- B_1 \dots B_m$ in L such that B and B' unify with mgu θ **then**
    assign fresh variables to $B :\!- B_1 \dots B_m$
    $R \leftarrow (R - B) \cup B_1 \cup \dots \cup B_m$
    apply θ to A and each element of R
  else
    exit while loop
  end if
end while

if $R = \varnothing$ **then**
  output A
else
  output *fail*
end if

To simulate this in the unification machine, given a program L we introduce the following transition rule for stacks in which the top is an *atom* rather than a *term*:

$$\frac{(\ S\,P(t_1,\ldots,t_m)\ ,\ \star\ )}{(\ S\,P(t_1,\ldots,t_m)\ ,\ \nu\vec{x}.\langle A\rangle.[A_n]\ldots[A_1]\ )} \tag{9}$$

corresponding to a non-deterministic choice of clause $A \coloneq A_1 \ldots A_n$ in L with free variables $\vec{x}$ such that $A = P(t'_1,\ldots,t'_m)$ unifies with $P(t_1,\ldots,t_m)$. Note that it doesn't matter what the rule is: Prolog would select only rules where $A = P(t'_1,\ldots,t'_m)$, but selecting another rule would immediately fail, so we can ignore that.

Proposition 3. *Given a query ?− A. with free variables $x_1,\ldots,x_n$ and program L,*

$$\frac{(\ x_1\ldots x_n\,A\ ,\ \star\ )}{\overline{(\ t_1\ldots t_n\ ,\ \star\ )}}$$

if and only if the abstract interpreter (with input ?− A. and L) outputs an instance of A with free variables $t_1,\ldots,t_n$.

Proof. By proposition 1, we have that

$$\frac{(\ S\,P(t_1,\ldots,t_m)\ ,\ \nu\vec{x}.\langle A\rangle.[A_n]\ldots[A_1]\ )}{\overline{(\ (S\,A_n\ldots A_1)[\theta]\ ,\ \star\ )}}$$

where θ is the mgu of $P(t_1,\ldots t_m)$ and A. We may therefore replace applications of (9) by a rule of the form:

$$\frac{(\ S\,P(t_1,\ldots,t_m)\ ,\ \star\ )}{(\ (S\,A_n\ldots A_1)[\theta]\ ,\ \star\ )} \tag{10}$$

where again $A \coloneq A_1 \ldots A_k$ is in L and θ is the mgu of $P(t_1,\ldots,t_m)$ and A. By induction, each state in the run of $(x_1\ldots x_n\,A,\star)$ must then be of the form $(y_1\ldots y_n\,B_m\ldots B_1,\star)$, and each step an instance of (10). Mapping each state $(y_1\ldots y_n\,B_m\ldots B_1,\star)$ to the resolvant set $R = \{B_m,\ldots,B_1\}$ gives a straightforward correspondence between machine steps and iterations of the while loop in an instance of the abstract interpreter with a fixed scheduling policy.

□

Proposed research

- Signatures modulo theory?
- Reduction on terms?
- Types?
- Confluence?