Second-order pattern unification

May 22, 2022

We show correctness of an algorithm computing the most general unifier of a list of term pairs involving binding operations and n-ary metavariables applied to distinct variables. We reason in the framework of Σ-monoids [4], but in the untyped case. In short, we show that a finite diagram in a second-order algebraic theory [7] generated by a binding signature has a limit as long as there exists a cone.

1 Related work

First-order unification was categorically described in [5]

[9] introduces pattern unification, a particular case of higher-order unification for the simply-typed lambda-calculus, where metavariables are applied to distinct variables. Hence, it has strong similarities with the problem we are studying. In particular, one can see our setting as a restricted case of theirs, since, in particular, we focus on second-order syntax. However, we can translate pattern unification problems in our setting extended to the simply-typed case, see Section 5, following ideas from [3].

[10] proposes a categorical understanding of pattern unification.

2 Notations

We denote by C^o the opposite category of C. In a category with coproducts, we denote by $in : A_i \hookrightarrow \coprod_i A_i$ the coproduct injection. If F is an endofunctor on a category C, we denote by μF or $\mu Z.F(Z)$ the initial F-algebra.

We denote by $\mathbb{F}$ the full subcategory of sets consisting of finite cardinals: objects are natural numbers and a morphism between n and m is a function $\{0, \ldots, n-1\} \to \{0, \ldots, m-1\}$.

Given a monad T on a category C, we denote the Kleisli category by Kl_T: objects are objects of C, and morphisms between c and c' are morphisms in C between c and Tc'. Note that there is a bijection between Kleisli morphisms $c \to Tc'$ and T-algebra morphisms $Tc \to Tc'$. We denote by f^* the T-algebra morphism induced by a Kleisli morphism. Composition of g and f is given by $g^* \circ f$. We denote by $I : \mathbb{N} \to Set$ the family of sets mapping n to the n^{th} cardinal set $\{0, \ldots, n-1\}$.

We denote metavariables by capital letters M, N,...

3 Setting

Let Σ be a binding signature. Then, there is a free Σ-monoid monad T on $Set^{\mathbb{N}}$ such that $T(X)$ is the syntax of Σ extended with an n-ary operation for each $x \in X_n$. More precisely, $T(X)_n$ is the set of terms whose free variables are in $\{0, \ldots, n-1\}$.

Example 1. Consider the case of λ-calculus. Then, $T(\emptyset)_n$ is the set of λ-terms t taking free variables in $\{0, \ldots, n-1\}$.

Consider $X \in Set^{\mathbb{N}}$ such that $X_2 = \{M\}$ and $X_{n\neq 2} = \emptyset$. Then $T(X)_n$ is the set of λ-terms t involving a metavariable M of arity 2 such that $fv(t) \subset \{0, \ldots, n-1\}$.

Consider $Y \in Set^{\mathbb{N}}$ such that $Y_2 = \{M, N\}$, $Y_0 = \{C\}$ and $Y_n = \emptyset$ otherwise. Then $T(Y)_n$ is the λ-term t with metavariables M,N of arity 2 and a constant metavariable C such that $fv(t) \subset \{0, \ldots, n-1\}$.

Note that metavariables are allowed to be applied to arbitrary terms here, not only (distinct) variables.

Definition 2. A morphism $X \to T(Y)$ is said *safe* if it factors through $T'(Y) \to T(Y)$, where $T'(Y)_n$ is the subset of $T(Y)_n$ consisting of terms that where metavariables are applied to distinct variables only, rather than arbitrary terms.

Remark 3. $T'(Y)$ is the initial algebra for $Z \mapsto I + Z \boxtimes I + \Sigma(Z)$, where

- we denote by Σ the endofunctor on $Set^{\mathbb{N}}$ corresponding to the binding signature Σ.
- $(Z \boxtimes I)_n = \coprod_p Z_p \times Inj(p, I_n)$. Note that this definition is not functorial in the second argument (unless we restrict to monomorphisms).

Notation 4. We denote by Kl_T the Kleisli category of the monad T: objects are families in $Set^{\mathbb{N}}$ and a morphism $X \to Y$ is a family morphism $X \to T(Y)$.

Remark 5. A (metavariable) substitution is precisely a morphism in the Kleisli category. For example, a Kleisli morphism from X to Y is a morphism $X \to T(Y)$, i.e., for each $n \in \mathbb{N}$, a map $X_n \to T(Y)_n$ which assigns a term taking free variables in $\{0, \ldots, n-1\}$ for each n-ary metavariable.

Remark 6. Consider the "operadic" monoidal product $A \boxtimes B_n = \coprod_p \coprod_{i_1+\cdots+i_p=n} A_p \times B_{i_1} \times \cdots \times B_{i_p}$ and Σ-monoids for this tensor products, with associated monad T'. In this setting, we think of $T'(X)_n$ as the set of terms withneed to exactly n free variables. Thus, Kleisli morphisms maps metavariables to terms using exactly once each argument.

Definition 7. A family $X \in Set^{\mathbb{N}}$ is said *finite* if $\coprod_n X_n$ is finite.

Remark 8. The full subcategory of Kl_T consisting of finite families corresponds to the second-order algebraic theory associated with T [7]. TODO: think over this remark.

Our main result consists in the following.

Theorem 9. *Let $V \rightrightarrows T(W)$ be a pair of safe morphisms between finite families V and W. If there is a coequalising Kleisli morphism, then there is coequaliser in $Kl(T)$.*

We use an explicit construction, detailed in the next section. This theorem allows to compute the most general unifier of two terms.

Corollary 10. *Let $t, u \in T(V)_n$ for some finite family $V \in Set^{\mathbb{N}}$ (thought as a specification of metavariables). If there exists a substitution unifying t and u, then there exists a most general unifier, i.e., a substitution $V \to T(W)$ such that any other unifying substitution uniquely factors through it.*

Proof. Giving two terms $t, u \in T(V)_n$ amounts to giving two parallel morphisms $yn \rightrightarrows T(V)$, where yn denotes the family defined by $yn_p = \emptyset$ if $p \neq n$ and yn_n is a singleton set. The most general unifier, if it exists, is the coequaliser of t and u. □

More generally, we can compute any finite colimit under the same condition.

Corollary 11. *Let $J : D \to Kl_T$ be a finite diagram selecting safe Kleisli morphisms and finite families. If there is a cocone, then J has a colimit.*

Proof. This follows from the computation of colimits as coequalisers and coproducts [6, Theorem V.2.2]. □

Let us finish this section by introducing some useful definitions, notations, and lemmas.

Lemma 12. *Free T-algebras extend to functors $\mathbb{F} \to Set$ preserving pullbacks, where $\mathbb{F}$ is the full subcategory of sets consisting in finite cardinals. Action on morphisms is given by renaming. Moreover, Kleisli morphisms are compatible with renaming.*

Proof. See Appendix A. □

Notation 13. If $n \in \mathbb{N}$, we denote the n^{th} cardinal set $\{0, \dots, n-1\}$ by n.

If $n \in \mathbb{N}$, we denote by yn the yoneda embedding into $Set^{\mathbb{N}}$, i.e., $yn(p)$ is empty if $n \neq p$ and a singleton set otherwise. We call *representable* any family which is isomorphic to some yn.

$I \in Set^{\mathbb{N}}$ denotes the family $I_n = n$.

Lemma 14. *Any family $X \in Set^{\mathbb{N}}$ is isomorphic to a coproduct of representable families. A family X is finite if and only if such a coproduct is finite.*

Proof. Clearly, any family X is isomorphic to $\coprod_n X_n yn \simeq \coprod_n \coprod_{x \in X_n} yn$. □

Notation 15. We represent a finite family $X \in Set^{\mathbb{N}}$ as a finite (metavariable) context $x_1 : n_1, \dots, x_p : n_p$ where $x_i \in X_{n_i}$.

4 Algorithm

The algorithm takes as input

- a (finite) metavariable context $\Gamma \in Set^{\mathbb{N}}$, that we denote by a list $M_1 : m_1, \dots M_n : m_n$ of metavariable symbols M_i with their associated arities m_i;
- a list of term pairs $t_i =_{n_i} u_i$, where n_i that t_i and u_i takes free variables in $\{0, \dots, n_i - 1\}$

It outputs:

- a metavariable context Δ
- a Kleisli map $\sigma : \Gamma \to T(\Delta)$, that is a substitution assigning to each metavariable $M : m$ declared in Γ a term with m free variables involving metavariables Δ.

In this section we inductively specify the algorithm through the judgement

$$\Gamma \vdash t_1 =_{n_1} u_1, \dots t_p =_{n_p} u_p \Rightarrow \sigma \dashv \Delta$$

that we sometimes abbreviate as

$$\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$$

The completeness statement (proven by induction) is the following.

Proposition 16. *Given a list of safe term pairs $\vec{t} = \vec{u}$ taking metavariables in Γ, if there exists a (finite) derivation tree of $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$, then σ is the most general unifier. Moreover such a derivation tree exists as long as there exists at least one unifier.*

We will justify this proposition by showing that each introduced rule is sound, in the sense that that it supports the induction step. We omit the proof that if the conclusion involves safe terms, the also do the premises.

At most one rule is applyable given an input $\vec{t} = \vec{u}$.

Proposition 17. *There is at most one derivation tree of $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$.*

Let us list indeed the rules according to the input $\vec{t} = \vec{u}$ in the conclusion:

- Empty list $\vec{t} = \vec{u} = ()$, in Section 4.1;
- Non empty, non singleton lists, $t_0 = u_0, \vec{t} = \vec{u}$, in Section 4.2;
- $M(\vec{x}) = N(\vec{y})$, in Section 4.3;
- $o(\vec{t}) = o(\vec{u})$ and $x = y$ in Section 4.4;
- $M(\vec{x}) = o(\vec{t})$, in Section 4.5;

- $M(\vec{x}) = x_i$, in Section 4.6.

Let us mention the missing cases, which can never be unified anyway.

- $o(\vec{t}) = o'(\vec{u})$ when $o \neq o'$;
- $M(\vec{x}) = y$ when $y \notin \vec{x}$
- $o(\vec{t}) = x$

4.1 Empty list

$$\frac{}{\Gamma \vdash () \Rightarrow id \dashv \Gamma}$$

Soundness is straightforward.

4.2 Stepwise construction

We can restrict to the case of a single coequaliser.

$$\frac{\Gamma \vdash t_0 =_{n_0} u_0 \Rightarrow \sigma_0 \dashv \Delta_1 \qquad \Delta_1 \vdash \vec{t}[\sigma_0] =_{\vec{n}} \vec{u}[\sigma_0] \Rightarrow \sigma \dashv \Delta_2 \qquad \vec{t} \text{ is not empty}}{\Gamma \vdash t_0 =_{n_0} u_0, \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \circ \sigma_0 \dashv \Delta_2}$$

Soundness of this rule follows from the following general categorical lemma.

Lemma 18. *Let $f_1, g_1 : A_1 \to B$ and $f_2, g_2 : A_2 \to B$ be morphisms in some category. Then the coequaliser of the induced parallel morphism $A_1 \coprod A_2 \rightrightarrows B$ is the morphism $B \to C \to D$ defined as follows (assuming the involved coequalisers exist):*

1. *$B \to C$ is the coequaliser of $A_1 \rightrightarrows B$;*
2. *$C \to D$ is the coequaliser of $A_2 \rightrightarrows B \to C$.*

Corollary 19. *The above rule is sound.*

Proof. Assume there is a common unifier for $t_0 = u_0, \vec{t} = \vec{u}$. By induction hypothesis, the premises are derivable and thus the rule can be applied. Moreover, again by induction hypothesis, the premises produce most general unifiers. The output substitution of the conclusion is also the most general unifier, thanks to the previous lemma by specialising it to the Kleisli category Kl_T, taking $A_1 = yn_0$ and $A_2 = \coprod_i yn_i$. □

4.3 Case $M(x_1, \ldots, x_m) =_q N(y_1, \ldots, y_n)$

We need to make the distinction whether $M = N$ or not.

$$\frac{(i_1, \ldots, i_p) \text{ is the family of common positions: } x_{\vec{i}} = y_{\vec{i}}}{\Gamma \vdash M(\vec{x}) =_q M(\vec{y}) \Rightarrow M(1, \ldots, m) \mapsto N(i_1, \ldots i_p) \dashv \Gamma \backslash \{M\}, N : p}$$

The premise states that $i : p \to m$ is the equaliser of $\vec{x}, \vec{y} : m \to q$ in $\mathbb{F}$.

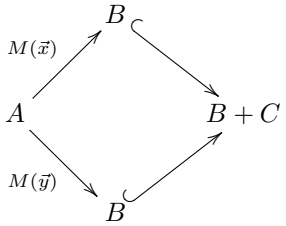
$$\frac{M \neq N \qquad (i_1, \ldots, i_p) \text{ and } (j_1, \ldots, j_p) \text{ are maximal such that } x_{\vec{i}} = y_{\vec{j}}}{\Gamma \vdash M(\vec{x}) =_q N(\vec{y}) \Rightarrow M(1, \ldots, m) \mapsto O(i_1, \ldots i_p), N(1, \ldots, n) \mapsto O(j_1, \ldots, j_p) \dashv \Gamma \backslash \{M, N\}, O : p}$$

The premise states that $m \xleftarrow{i} p \xrightarrow{j} n$ is a pullback of $m \xrightarrow{\vec{x}} q \xleftarrow{\vec{y}} n$ in $\mathbb{F}$.

Proposition 20. *The above rules are sound.*

Proof. We prove that the output substitution is the most general unifier.

Let us consider the first rule. We decompose Γ as $M : m, \Gamma'$ the coproduct of ym and Γ'. The term $M(\vec{x})$ corresponds to the composition of $yq \xrightarrow{M(\vec{x})} T(ym) \hookrightarrow T(\Gamma)$. More abstractly, we want to compute the coequaliser, in Kl_T, of



for $A = yq$, $B = ym$, $C = \Gamma'$. It is enough to compute the coequaliser of $f : B \to D$ of $A \rightrightarrows B$, for the desired coequaliser is then $B + C \xrightarrow{f+C} D + C$. So we need to compute the coequaliser of $yq \rightrightarrows T(ym)$. As we explain in Section A, the functor $\mathbb{N} \xrightarrow{y} Set^{\mathbb{N}} \to Kl_T$ extends to a functor $\mathbf{y} : \mathbb{F} \to Kl_T^{op}$ such that the coequaliser is precisely that of $\mathbf{y}\vec{x}$ and $\mathbf{y}\vec{y}$. Since $\mathbf{y}$ preserves equalisers (Lemma 37), the coequaliser is $ym \xrightarrow{\mathbf{y}i} T(yp)$, where $i : p \to m$ is the equaliser of $\vec{x}$ and $\vec{y}$.

The second rule can be justified similarly. □

4.4 Case $o(\vec{t}) = o(\vec{u})$ and $x = y$

$$\frac{o \text{ has binding arity } \vec{n} \qquad \Gamma \vdash \vec{t} =_{q+\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta}{\Gamma \vdash o(\vec{t}) =_q o(\vec{u}) \Rightarrow \sigma \dashv \Delta}$$

Lemma 21. *This rule is sound.*

Proof. The result follows from the fact that a substitution unifies $o(\vec{t}) = o(\vec{u})$ if and only if it unifies $\vec{t}$ with $\vec{u}$. □

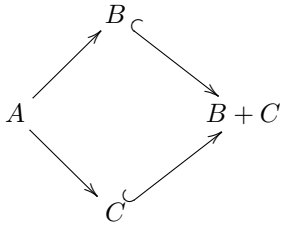
$$\overline{\Gamma \vdash x =_q x \Rightarrow id \dashv \Gamma}$$

4.5 Case $M(\vec{x}) = o(\vec{u})$

$$\frac{o \text{ has binding arity } (n_1, \dots, n_p) \qquad M \text{ does not occur in } \vec{u} \qquad \Gamma\backslash\{M\}, \vec{N} : m + \vec{n} \vdash N_1(0, \dots, n_1 - 1, x_1 + n_1, \dots, x_m + n_1) =_{q+n_1} u_1, \cdots \Rightarrow \sigma \dashv \Delta}{\Gamma \vdash M(\vec{x}) =_q o(\vec{u}) \Rightarrow \sigma \circ M(1, \dots m) \mapsto o(\vec{N}) \dashv \Delta}$$

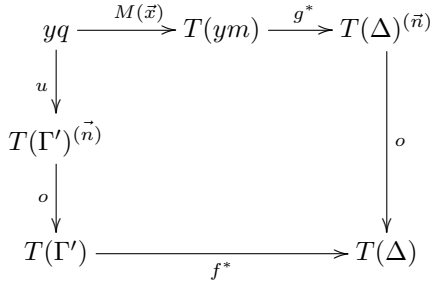
Lemma 22. *The above rule are sound.*

Proof. If there is a unifier, then M can't appear in $\vec{u}$ for size reason. Let us write Γ as $M : m, \Gamma'$, the coproduct of ym and Γ'. The term $M(\vec{x})$ corresponds to the composition of $yq \to T(ym)$ selecting $M(\vec{x})$ with the coproduct injection $T(ym) \hookrightarrow T(\Gamma)$. As M does not appear in $o(\vec{u})$, the corresponding morphism $yq \to T(\Gamma)$ factors through the coproduct injection $T(\Gamma') \hookrightarrow T(\Gamma)$. In other words, we want to compute the coequaliser (in Kl_T) of



with $A = yq$, $B = ym$ and $C = \Gamma'$. This is equivalent to the pushout of $B \leftarrow A \to C$. The morphism $A \to C$ corresponding to $o(\vec{u})$ factors as $yq \xrightarrow{u} T(\Gamma')^{(\vec{n})} \xrightarrow{o} T(\Gamma')$.

Now, let us study the category of unifiers. A unifier $ym \to T(\Delta) \leftarrow \Gamma'$ must maps M to some $o(\dots)$, so that $ym \to T(\Delta)$ factors as $ym \to T(\Delta)^{(\vec{n})} \xrightarrow{o} T(\Delta)$. In other words, a unifier is a family Δ with morphisms $f : \Gamma' \to T(\Delta)$ and $g : ym \to T(\Delta)^{(\vec{n})}$ such that the following diagram commutes:



Now, since f^* is a T-algebra morphism, commutation of the previous diagram

is equivalent to commutation of the following one:

$$\begin{array}{ccccc} yq & \xrightarrow{M(\vec{x})} & T(ym) & \xrightarrow{g^*} & T(\Delta)^{(\vec{n})} \\ {\scriptstyle u}\downarrow & & & & \\ T(\Gamma')^{(\vec{n})} & & & & \downarrow{\scriptstyle o} \\ {\scriptstyle f^{*(\vec{n})}}\downarrow & & & & \\ T(\Delta)^{(\vec{n})} & & \xrightarrow[o]{} & & T(\Delta) \end{array}$$

Since o is injective, this is equivalent to commutation of the following diagram.

$$\begin{array}{ccc} yq & \xrightarrow{M(\vec{x})} & T(ym) \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle g^*} \\ T(\Gamma')^{(\vec{n})} & \xrightarrow[f^{*(\vec{n})}]{} & T(\Delta)^{(\vec{n})} \end{array}$$

Now, a morphism $u : yk \to Y^{(\vec{n})}$ is equivalently given by a morphism $u' : y(k+\vec{n}) \to Y$, where $y(k+\vec{n})$ denotes the coproduct $\coprod_i y(k+n_i)$. Then u can be retrieved from u' as the composition $yk \xrightarrow{d} y(k+\vec{n})^{(\vec{n})} \xrightarrow{u'^{(\vec{n})}} Y^{(\vec{n})}$. Thus, a unifier is given by a family Δ together with Kleisli morphisms $g' : y(m+\vec{n}) \to T(\Delta)$ and $h : \Gamma' \to T(\Delta)$ such that the following diagram commutes.

$$\begin{array}{ccc} y(q+\vec{n}) & \xrightarrow{M(\vec{x})'} & T(y(m+\vec{n})) \\ {\scriptstyle u'}\downarrow & & \downarrow{\scriptstyle g'^*} \\ T(\Gamma') & \xrightarrow[f^*]{} & T(\Delta) \end{array} \tag{1}$$

Thus, a unifier is a cocone over the pushout diagram

$$\begin{array}{ccc} y(q+\vec{n}) & \xrightarrow{M(\vec{x})'} & T(y(m+\vec{n})) \\ {\scriptstyle u'}\downarrow & & \\ T(\Gamma') & & \end{array}$$

The premise is precisely computing this colimit, producing $\sigma : \Gamma' + y(m+\vec{n}) \to T(\Delta)$:

$$\Gamma\backslash\{M\}, \vec{N} : m+\vec{n} \vdash N_1(0, \ldots, n_1-1, x_1+n_1, \ldots, x_m+n_1) =_{q+n_1} u_1, \cdots \Rightarrow \sigma \dashv \Delta$$

The desired unifier is then the composition of σ with the coproduct morphism (in Kl_T) $\Gamma' + ym \xrightarrow{id_{\Gamma'}+v} T(\Gamma' + y(m+\vec{n}))$, where v is the composite $ym \to T(y(m+\vec{n}))^{(\vec{n})} \xrightarrow{o} T(y(m+\vec{n}))$. □

4.6 Case $M(\vec{x}) = x_i$

$$\frac{}{\Gamma \vdash M(\vec{x}) =_q x_i : M(1, \dots, m) \mapsto i \dashv \Gamma \backslash \{M\}}$$

A symmetric rule must be introduced as well.

4.7 Termination

This section is devoted to the proof of the following proposition.

Proposition 23. *There is no infinite derivation tree of* $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$.

The main difficulty to show termination is that the case $M(\vec{x}) = o(\vec{u})$ involves a recursive call with an extended context, while all the other rules are reducing or keeping the same context size.

It turns out that after encountering the case $M(\vec{x}) = o(\vec{u})$, the algorithm virtually enters a "pruning" phase where the left handsides will always be distinct metavariables that don't appear in the right handsides. We say that the unification problem is *directed* if it is of this shape.

Lemma 24. *If there is a finite derivation of* $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$ *where* $\vec{t} = \vec{u}$ *is directed, then all the involved nodes are directed..*

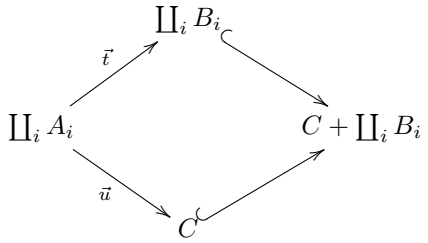
The difficult case is the stepwise rule. It relies on the following lemma.

Lemma 25. *Let* $\Gamma \vdash t_0, \vec{t} = u_0, \vec{u}$ *be a directed unification problem, and assume given a finite derivation tree* $\Gamma \vdash t_0 = u_0 \Rightarrow \sigma \dashv \Delta$. *Then,* $\vec{t}[\sigma] = \vec{u}[\sigma]$ *is directed.*

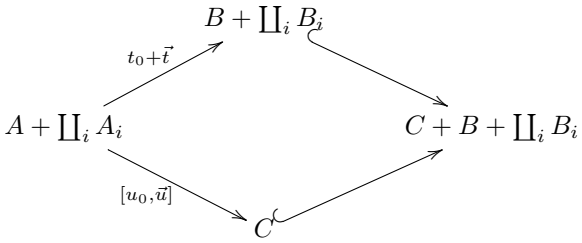
This in turn requires the following lemma proved by induction.

Lemma 26. *Assume there is a finite derivation of* $\Gamma, \Gamma' \vdash \vec{t} = \vec{u} \Rightarrow \sigma \dashv \Delta$, *such that metavariables in* $\vec{t}$,$\vec{u}$ *are in* Γ. *Then, there exists* Δ' *such that* $\Delta \simeq \Delta', \Gamma'$ *and* σ *through this isomorphism, is of the shape* $\sigma' + id_{\Gamma'}$. *In other words, the output substitution preserves any unused metavariable and never use them for instantiation otherwise.*

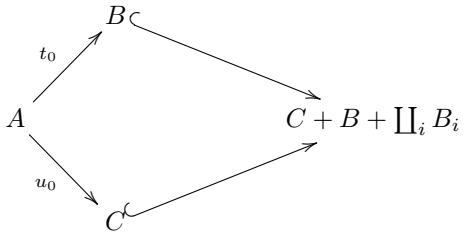
Proof of Lemma 25. Note that a directed unification problem $\vec{t} = \vec{u}$ corresponds to a diagram in Kl_T of the shape



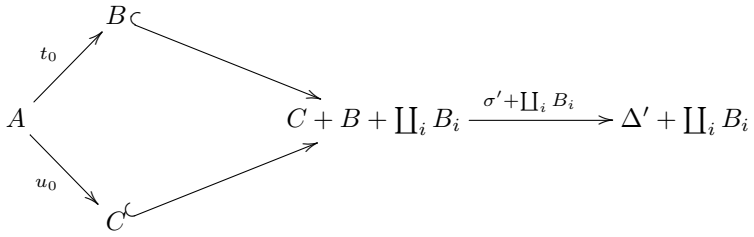
In particular, a directed unification problem $t_0, \vec{t} = u_0, \vec{u}$ has this shape:



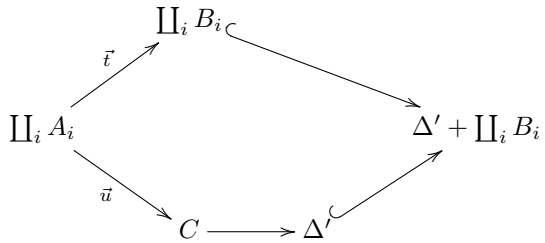
The most general unifier of t_o and u_0 corresponds to the coequaliser of



By Lemma 26, the coequaliser computed by the derivation of $\Gamma \vdash t_0 = u_0 \Rightarrow \sigma \dashv \Delta$ has the shape



Now, $\vec{t}[\sigma] = \vec{u}[\sigma]$ corresponds to the diagram



which is indeed directed. □

Now we can prove easily

Lemma 27. *If there is a finite derivation tree of* $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$*, then*

- $|\Gamma| \geq |\Delta|$

- *if* $|\Gamma| = |\Delta|$*, then* σ *is a renaming, in the sense that it instantiates metavariables with metavariables.*
- *if* $\vec{t} = \vec{u}$ *is directed, then* $|\Gamma| = |\Delta| + |\vec{t}|$*, where* $|\vec{t}|$ *is the length of* $\vec{t}$*, as a list.*

Finally, to prove termination, we consider the lexicographic order induced by the context size and the size of the involved terms to each judgement $\Gamma \vdash \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \dashv \Delta$. More precisely,

- the context size is defined as the length of Γ if it is not directed, or as the total number of metavariables involved in the right handsides otherwise.
- the size of a non empty term list $||\vec{t}||$ is defined as $|\vec{t}| - 1 + \Sigma_i ||t_i||$
- the size of a term $||o(\vec{t})||$ is defined as $1 + ||\vec{t}||$, and $||M|| = ||x|| = 0$.

It is easy to show that each recursive call is decreasing with respect to this order. The crucial point, in the stepwise rule, is the following lemma.

Lemma 28. *If* σ *is a renaming, then* $||t[\sigma]|| = ||t||$.

5 Higher-order pattern unification

In this section, we recall higher-order pattern unification introduced in [9] and discuss the differences with our setting. We also sketch how to encode pattern unification problems in our setting extended to the simply-typed case. It would be nice to compare our categorical proof with [10].

Higher-order pattern unification focuses on unifying a pair of closed simply-typed lambda-terms with metavariables applied to distinct variables. Lambda-terms are considered in β-short η-long normal forms, except for metavariables arguments which are not η-expanded (they are metavariables). Roughly, the syntax of terms goes as follows.

$$t ::= \lambda(x : \tau).t | x\vec{t} | M\vec{x}$$

with a standard typing judgement, with types generated from a set of base types B typically denoted by b (we mix metavariables and free variables in the same context Γ for concision):

$$\frac{x : \vec{\tau} \Rightarrow b \in \Gamma \quad \Gamma \vdash \vec{t} : \vec{\tau}}{\Gamma \vdash x\vec{t}} \qquad \frac{M : \vec{\tau} \Rightarrow b \in \Gamma \quad \vec{x} : \vec{\tau} \in \Gamma}{\Gamma \vdash M\vec{t}} \qquad \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda(x : \tau).t : \tau \Rightarrow \tau'}$$

On the surface, our setting looks quite different in may respects:

- we do not focus on simply-typed lambda calculus quotiented by $\beta\eta$-equivalence, but on any second-order syntax specified by a binding signature Σ, without any equation;
- we work in the untyped case.

- our unification problem can involve open terms.

If we forget about the types, we can translate unification problems back and forth, as we show in the following sections.

Let us first note that the syntax of normal terms clearly embeds into untyped lambda-calculus without $\beta\eta$-equivalence, as specified by a binding signature, and so our unification algorithm applies. What we should additionnally show is that the output substitution preserves typing and do not creates any redex.

5.1 Encoding our unification problems

We do the case of a single term pair $\Gamma \vdash t =_q u$. By λ-abstracting over the metavariable context and the q free variables, we get a higher-order pattern unification problem $\lambda\Gamma\lambda 0..\lambda(q-1).t = \lambda\Gamma\lambda 0..\lambda(q-1).u$.

5.2 Encoding pattern unification problems

Clearly the previous encoding only generate pattern unification problem involving at most second-order types (for metavariables).

However, it turns out that the syntax of simply-typed λ-calculus in β-short η-long form can be described by a simply-typed binding signature. The idea is that for each simple type $\vec{\tau} \Rightarrow b$, we have an application operation $-@- : (\vec{\tau} \Rightarrow b) \times [\vec{\tau}] \to b$, where $[\vec{\tau}] := [\tau_1] \times \ldots [\tau_n]$ and $[\vec{\tau} \Rightarrow b] := b^{\vec{\tau}}$. Let T be the (finitary) monad on Set^S generated by this signature, where S is the set of simple types. Then, the set of closed λ-terms of type $\vec{\tau} \Rightarrow b$ is isomorphic to $T(\vec{\tau})_b$, where $\vec{\tau}$ is thought of as a family $S \to Set$. TODO find references.

5.3 Comparison with [10]

Notes: in the simply typed setting with set of types S, the free Σ-monoid monad acts on $[I/S \times S, Set]$, where $I : \mathbb{N} \to Set$ is the canonical inclusion. Or $[\mathbb{N}_f^S \times S, Set]$ where the subscript f means we restrict to finite families.

Let Λ be the monad on $[\mathbb{N}_f^S \times S, Set]$ generating λ-terms modulo β and η with metavariables.

Pat is $\mathbb{F}^S_{inj,f}$. The category of substitutions Sub is again the second-order algebraic theory of Λ.

$Tm(\Gamma, \Delta, \tau)$ is $\Lambda(\Gamma)_{\Delta,\tau}$. A term is a $y(\Gamma, \Delta, \tau) \to Tm(\Gamma, \Delta, \tau)$. Ou alors, $y(\Delta, \tau) \to Tm(\Gamma, -, -)$ et la on se rapproche de mon idee. TODO: prouver que les conversions preservent la condition.

References

[1] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015.

[2] Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proc. 35th ACM/IEEE Symposium on Logic in Computer Science* ACM, 2020.

[3] James Cheney. Relating nominal and higher-order pattern unification. In *Proceedings of the 19th international workshop on Unification (UNIF 2005)*, pages 104–119. LORIA research report A05, 2005.

[4] Marcelo Fiore and Dmitrij Szamozvancev. Formal metatheory of second-order abstract syntax. *Proceedings of the ACM on Programming Languages*, 6(POPL), 2022.

[5] Joseph A. Goguen. What is unification? - a categorical view of substitution, equation and solution. In *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic, 1989.

[6] Saunders Mac Lane. *Categories for the Working Mathematician.* Number 5 in Graduate Texts in Mathematics. Springer, 2nd edition, 1998.

[7] Ola Mahmoud. Second-order algebraic theories. Technical Report UCAM-CL-TR-807, University of Cambridge, Computer Laboratory, October 2011.

[8] Conor McBride. First-order unification by structural recursion. *J. Funct. Program.*, 13(6):1061–1075, 2003.

[9] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[10] Andrea Vezzosi and Andreas Abel. A categorical perspective on pattern unification. *RISC-Linz*, page 69, 2014.

A Free T-algebras preserve pullbacks

In this section we show that renaming turn free T-algebras into functors $\mathbb{F} \to Set$ preserving pullbacks. In fact, we are going to show it for the initial T-algebra, since a free T-algebra on $X \in Set^{\mathbb{N}}$ is equivalently the initial Σ'-monoid, for Σ' the binding signature extending Σ with first order operations as specified by X.

Let us first explain where renaming comes from. Every binding signature Σ induces an endofunctor on $Set^{\mathbb{N}}$ that we still denote by Σ. For example, in the case of λ-calculus, $\Sigma(X)_n = X_n \times X_n + X_{n+1}$.

Lemma 29. *Denoting Σ^* the free monad $\Sigma^* X = \mu Z.X + \Sigma(Z)$ induced by a binding signature Σ, the initial Σ-monoid is $\Sigma^* I$.*

Proof. See [2, Theorem 2.15]. □

Lemma 30. *Given a binding signature Σ, the free monad Σ^* lifts to the category $Set^{\mathbb{F}}$. Denoting $\underline{I}$ the canonical embedding $\mathbb{F} \to Set$, renaming on $\Sigma^* I$ is given by the renaming structure on $\Sigma^* \underline{I}$.*

Proof. Renaming on $\Sigma^* I$ is induced by substitution. Substitution on $\Sigma^* I$ induced by a monoid structure on them (for a monoidal structure on $Set^{\mathbb{F}}$ or $Set^{\mathbb{N}}$, as in [1]) induced by a strength $\Sigma(A) \otimes B \to \Sigma(A \otimes B)$.

TODO □

Lemma 31. *The previous lemma induces a functor $\mathcal{U} : Kl_T \to Set^{\mathbb{F}}$.*

What remains to show is that this renaming preserves pullbacks. Let us reason on the category $Set^{\mathbb{F}}$ on which Σ^* lifts, as we just argued. Now, $\Sigma^* I$ is computed as the initial algebra of $H : Set^{\mathbb{F}} \to Set^{\mathbb{F}}$ mapping X to $I + \Sigma(X)$. This means that $\Sigma^* I$ is the colimit over the initial ω-chain

$$\emptyset \to H(\emptyset) \to \cdots \to H..H(..(\emptyset..)) \to \ldots$$

Since pullback preserving functors are preserved by filtered colimits (as filtered colimits commute with finite limits), it is enough to show that each functor $H^{\circ n}(\emptyset)$ preseves pullbacks, which again amounts to showing that H preserves functors preservings pullbacks.

Lemma 32. *The endofunctor $H : Set^{\mathbb{F}} \to Set^{\mathbb{F}}$ maps functors preserving pullbacks to functors preserving pullbacks.*

Proof. Let F be a functor $\mathbb{F} \to Set$ preserving pullbacks. Let

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & & \downarrow \\ C & \longrightarrow & D \end{array}$$

be a pullback in $\mathbb{F}$. Let us show that the following square is a pullback.

$$\begin{array}{ccc} H(F)_A & \longrightarrow & H(F)_B \\ \downarrow & & \downarrow \\ H(F)_C & \longrightarrow & H(F)_D \end{array}$$

Since pullbacks commute with coproducts in Set, this is enough to show that the following square is a pullback

$$\begin{array}{ccc} \Sigma(F)_A & \longrightarrow & \Sigma(F)_B \\ \downarrow & & \downarrow \\ \Sigma(F)_C & \longrightarrow & \Sigma(F)_D \end{array}$$

This follows from the fact that $\Sigma(F)$ is isomorphic to some $\coprod_i F^{(\vec{n_i})}$, and coproducts commute with connected limits. □

Notation 33. Let $Set_p^{\mathbb{F}}$ be the full subcategory of functors preserving pullbacks.

Lemma 34. *The functor* $\mathbb{N} \xrightarrow{y} Set^{\mathbb{N}} \to Kl_T$ *extends to a functor* $\mathbf{y} : \mathbb{F}^o \to Kl_T$ *such that* $Kl_T(\mathbf{y}m, X) \simeq Set_p^{\mathbb{F}}(Jm, TX)$ *is natural in* m.

Proof. The isomorphism $Kl_T(ym, X) \simeq TX_m \simeq Set^{\mathbb{F}}(Jm, TX)$ means that the following diagram commutes up to isomorphism

$$\begin{array}{ccccccc}
\mathbb{N} & \xrightarrow{y} & (Set^{\mathbb{N}})^o & & \longrightarrow & & Kl_T^o \\
\downarrow & & & & & & \downarrow{\scriptstyle y} \\
\mathbb{F} & \xrightarrow[y]{} & (Set^{\mathbb{F}})^o & \xrightarrow[y]{} & Set^{Set^{\mathbb{F}}} & \xrightarrow[Set^{\mathcal{U}}]{} & Set^{Kl_T}
\end{array}$$

where y denotes the yoneda embedding. Since the left vertical morphism is bijective on objects and the right vertical morphism is full and faithful, there is a filling of this square $\mathbf{y} : \mathbb{F}^o \to Kl_T$ making triangles both commute up to isomorphism (TODO: find references) □

Remark 35. Another way to define $\mathbf{y}$ goes as follows. There is a Σ-monoid monad T' on $Set^{\mathbb{F}}$ and T is actually isomorphic to $RT'L$, where L is the left Kan extension and R its right adjoint (precomposition by $\mathbb{N} \to \mathbb{F}$). As a consequence, there is a fully faithful Kleisli extension

$$\begin{array}{ccc}
Set^{\mathbb{N}} & \xrightarrow{L} & Set^{\mathbb{F}} \\
\downarrow & & \downarrow \\
Kl_T & \longrightarrow & Kl_{T'}
\end{array}$$

Then, we can define $\mathbf{y}$ as filling the square

$$\begin{array}{ccc}
\mathbb{N} & \longrightarrow & Kl_T^o \\
\downarrow & & \downarrow \\
\mathbb{F} & \longrightarrow & Kl_{T'}^o
\end{array}$$

Remark 36. Given a morphism $f : m \to n$ in $\mathbb{F}$, the induced morphism $\mathbf{y}f : yn \to T(ym)$ selects the term $t \in T(ym)_n$ defined as renaming the canonical term $M(0, \ldots, m-1) \in T(ym)_m$, corresponding to the unit $ym \to T(ym)$, along f. In other words, $\mathbf{y}f$ selects the term $M(f_0, \ldots f_{m-1})$.

Lemma 37. $\mathbf{y}^o : \mathbb{F} \to Kl_T^o$ *preserves pullbacks, hence finite connected limits.*

Proof. We have a natural isomorphism $Kl_T(\mathbf{y}m, X) \simeq Set_p^{\mathbb{F}}(Jm, TX)$, where J is the yoneda embedding in $\mathbb{F}^o \to Set^{\mathbb{F}}$: $Jn_m = m^n$. It follows that $\mathbf{y}$ preserves any colimit that J preserves. Now, since J is the yoneda embedding, J preserves pullbacks, as we restrict to the category of presheaves preserving pullbacks. □

Corollary 38. *Let $f : p \to q$ be an injective map. Then, the induced map $\mathbf{y}f : yq \to T(yp)$ is epimorphic.*

Proof. The proof follows from Lemma 37 by noting that $f : A \to B$ is monomorphic if and only if the following square is a pullback.

$$\begin{array}{ccc} A & \xrightarrow{id} & A \\ {\scriptstyle id}\downarrow & & \downarrow{\scriptstyle f} \\ A & \xrightarrow[f]{} & B \end{array}$$

□

Remark 39. Such an epimorphism may not be induce a surjection $\mathbf{y}f^* : T(yq) \to T(yp)$, even when. For example, consider the initial map $0 \to 1$, inducing $T(y1) \to T(y0)$, morally mapping the unary metavariable $M(x)$ to the constant metavariable N. If there is closed term, then $N \in T(y0)_0$ has no preimage.

B Separating the pruning phase

(WIP) Here we present another algorithm which does justice to the so-called pruning phase mentioned in Section 4.7. The benefit is that termination is easier to prove, at the cost of three more rules to prove correctness of.

We introduce a new *pruning* judgement (as usual, the arrows mean lists)

$$\Gamma \vdash \vec{x_1} \supset_{q_1} t_1, \ldots, \vec{x_n} \supset_{q_n} t_n \Rightarrow \sigma \dashv \Delta$$

abbreviated as

$$\Gamma \vdash \overrightarrow{\vec{x} \supset_q t} \Rightarrow \sigma \dashv \Delta$$

where

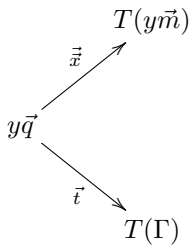
- $\vec{x}$ is a list of distinct variables in q
- t is a term in $T\Gamma_q$
- $\sigma : \Gamma \to T\Delta$ is a renaming (it replaces metavariables with metavariables).

The idea is that σ reduce the arguments of the metavariables in t so that they don't mention any variable outside $\vec{x}$.

Let us give the correctness statement before giving the rules

Proposition 40. *Given $\Gamma \vdash \overrightarrow{\vec{x} \supset_q t}$, there exists exactly one σ and one Δ such that there exists a finite derivation tree of $\Gamma \vdash \overrightarrow{\vec{x} \supset_q t} \Rightarrow \sigma \dashv \Delta$. Moreover, there*

exists $u : y\vec{m} \to T(\Delta)$ such that u and σ is colimiting for the pushout diagram



Note that such a u is unique, by Lemma 38 (indeed, it corresponds to $\vec{t}$ restricted to smaller free variable contexts).

The main point is that we can replace a few rules (three of them, in fact: $M = o(\vec{u})/x_i/N$) with the following one:

$$\frac{\begin{array}{c}M \text{ does not occur in } t \\ \Gamma\backslash\{M\} \vdash \vec{x} =_q t \Rightarrow \sigma \dashv \Delta\end{array}}{\Gamma \vdash M(\vec{x}) =_q t \Rightarrow \sigma, M(1, \dots m) \mapsto t[\sigma] \dashv \Delta}$$

Note that we know a posteriori that $t[\sigma]$ takes free variables in $1, \dots m$ by Proposition 40. To be more rigorous, we could make the algorithm outputs the u mentionned in Proposition 40 as well.

From the original algorithm, we need to keep the stepwise/empty rule, the cases $M(\vec{x}) = M(\vec{y})$ and $o(\vec{t}) = o(\vec{u})$.

We have the empty rule and the following stepwise rule

$$\frac{\Gamma \vdash \vec{x_0} =_{n_0} t_0 \Rightarrow \sigma_0 \dashv \Delta_1 \qquad \Delta_1 \vdash \overrightarrow{\vec{x} =_q t[\sigma_0]} \Rightarrow \sigma \dashv \Delta_2 \qquad \vec{t} \text{ is not empty}}{\Gamma \vdash t_0 =_{n_0} u_0, \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma \circ \sigma_0 \dashv \Delta_2}$$

Then, we have the cases $M = o(\vec{u})/x_i/N$, that we detail below. So, this version adds three more rules compared to the original algorithm.

$$\frac{}{\Gamma \vdash \vec{x} =_q x_i : id \dashv \Gamma}$$

$$\frac{\begin{array}{c}o \text{ has binding arity } (n_1, \dots, n_p) \\ \Gamma \vdash 0, \dots, n_1 - 1, \vec{x} + n_1 =_{q+n_1} t_1, \cdots \Rightarrow \sigma \dashv \Delta\end{array}}{\Gamma \vdash \vec{x} =_q o(\vec{t}) \Rightarrow \sigma \dashv \Delta}$$

$$\frac{(i_1, \dots, i_p) \text{ and } (j_1, \dots, j_p) \text{ are maximal such that } x_{\vec{i}} = y_{\vec{j}}}{\Gamma \vdash \vec{x} =_q N(\vec{y}) \Rightarrow N(1, \dots, n) \mapsto O(j_1, \dots, j_p) \dashv \Gamma\backslash\{N\}, O : p}$$

Since the pruning judgement is standalone (it never switches to the non-directed judgement), we can prove its termination first. This is relatively easy,

because the size of terms in the premises of each rule decrease (observing, in the stepwise rule, that renaming preserves the size). Then, to prove termination of the non-directed judgement, we note that the directed judgement preserves the size of the context. Then, the non-directed judgement either preserves the size of the context and ouputs a renaming, either strictly reduces its size.

C A structurally tail-recursive algorithm

WIP

The stepwise rules make the algorithm not tail-recursive. However, by taking an additional substitution as input as in [8], we can turn it into a tail-recursive algorithm. Moreover, in our case, this moreover enables structural recursion, by delaying the effects of renaming to the leaves.