Generic pattern unification: a categorical approach

Ambroise Lafont[0000−0002−9299−641X]

University of Cambridge

Abstract. We give a generic correctness proof for pattern unification in a categorical setting. The syntax with metavariables is generated by a free monad applied to finite coproducts of representable presheaves, and the most general unifier is computed as a coequaliser in the Kleisli category of this monad. Beyond simply typed syntax with binders, our categorical proof handles unification for linear or (intrinsic) polymorphic syntax such as system F.

Keywords: Unification · Category theory.

1 Introduction

We first provide a short introduction to pattern unification in Section 1.1, and then give some intuition for our categorical generalisation, in Section 1.2.

1.1 A short introduction to pattern unification

Unification consists in finding the most general unifier of two terms involving metavariables. To be more explicit, a unifier is a substitution that replaces metavariables with terms, potentially involving metavariables, such that the two substituted terms are equal. The most general unifier is the one that uniquely factors any other unifier.

In pattern unification, the arguments of a metavariable are restricted to be distinct variables. In that case, we can design an algorithm that either fails in case there is no unifier, either computes the most general unifier. Roughly, the algorithm recursively inspect the structure of the given pair of terms, until reaching a metavariable application $M(x_1, \dots, x_n)$ at top level. It then performs an *occur-check*, checking whether this metavariable M occurs in the other handside u. If not, it enters a *pruning phase*, removing all *outbound* variables in u, i.e., variables that are not among $x_1, \dots, x_n$. It does so by producing a substitution that reduces the arities of the metavariables occuring in u, so that they are not applied to any outbound variable after substitution.

If the occur-check is not successful, meaning that M appears in u, then there is no unifier unless M appears at top-level, because the size of substituted terms can never match. If M indeed appears at the top level in u, then the most general unifier replaces M with a new metavariable whose arity is the number of common variables positions in both handsides.

1.2 Generic pattern unification

In this section, we consider more formally pattern unification for the syntax of pure λ-calculus. This case study allows us to motivate our categorical account of pattern unification.

Consider the category of functors $[\mathbb{F}_m, \mathrm{Set}]$ from $\mathbb{F}_m$, the category of finite cardinals and injections between them, to the category of sets. A functor $X : \mathbb{F}_m \to \mathrm{Set}$ can be thought of as assigning to each natural number n a set X_n of expressions with free variables taken in the set $\underline{n} = \{0, \ldots, n-1\}$. The action on morphisms of $\mathbb{F}_m$ means that these expressions come equipped with injective renamings. Pure λ-calculus is such a functor Λ satisfying the recursive equation $\Lambda_n \cong \underline{n} + \Lambda_n \times \Lambda_n + \Lambda_{n+1}$, where $- + -$ is disjoint union.

In pattern unification, we consider extensions of this syntax with metavariables taking a list of distinct variables as arguments. As an example, let us add a metavariable of arity p. The extended syntax Λ' now satisfies the recursive equation $\Lambda'_n = n + \Lambda'_n \times \Lambda'_n + \Lambda_{n+1} + Inj(p, n)$, where $Inj(p, n)$ is the set of injections between the cardinal sets p and n, corresponding a choice of arguments for the metavariable. In other words, $Inj(p, n)$ is nothing but the set of morphisms between p and n in the category $\mathbb{F}_m$, which we denote by $\mathbb{F}_m(p, n)$.

Obviously, the functors Λ and Λ' satisfy similar recursive equations. Denoting Σ the endofunctor on $[\mathbb{F}_m, \mathrm{Set}]$ mapping F to $I + F \times F + F(- + 1)$, where I is the functor mapping n to the n^{th} cardinal set $\underline{n}$, the functor Λ can be characterised as the initial algebra for Σ, thus satisfying the recursive equation $\Lambda \cong \Sigma(\Lambda)$, while Λ' is characterised as the initial algebra for $\Sigma(-) + yp$, where yp is the representable functor $\mathbb{F}_m(p, -) : \mathbb{F}_m \to \mathrm{Set}$, thus satisfying the recursive equation $\Lambda' \cong \Sigma(\Lambda') + yp$. In other words, Λ' is the free Σ-algebra on yp. Therefore, denoting T the free Σ-algebra monad, Λ is $T(0)$ and Λ' is $T(yp)$. Similarly, if we want to extend the syntax with another metavariable of arity q, then the resulting functor would be $T(yp + yq)$.

In the view to abstracting pattern unification, these observations motivate considering functors categories $[\mathcal{A}, \mathrm{Set}]$, where $\mathcal{A}$ is a small category where all morphisms are monomorphic (to account for the pattern condition enforcing that metavariable arguments are distinct variables), together with an endofunctor Σ on it. Then, the abstract definition of a syntax extended with metavariables is the free Σ-algebra monad T applied to a finite coproduct of representable functors.

To understand how a unification problem is stated in this general setting, let us come back to the example of pure λ-calculus. Consider Kleisli morphisms for the monad T. A morphism $\sigma : yp \to T(yn)$ is equivalently given (by the Yoneda Lemma) by an element of $T(yn)_p$, that is, a λ-term t potentially involving a metavariable of arity n, with p free variables. Note that this is the necessary data to substitute a metavariable M of arity p: then, $M(x_1, \ldots, x_p)$ gets replaced with $t[i \mapsto x_{i+1}]$. Thus, Kleisli morphisms account for metavariable substitution and for term selection. Considering a pair of composable Kleisli morphism $yp \to T(yn)$ and $yn \to T(ym)$, if we interpret the first one as a term $t \in T(yn)_p$ and the second one as a metavariable substitution σ, then, the composition corresponds to the substituted term $t[\sigma]$. Now, a unification problem can be stated as pair of parallel Kleisli morphisms

$$\begin{tikzcd} yp \arrow[r, shift left] \arrow[r, shift right] & T(yq_1 + \dots + yq_n) \end{tikzcd}$$

corresponding to selecting a pair of terms with p free variables and involving metavariables of arity q_1, ..., q_n. A unifier is nothing but a Kleisli morphism coequalising this pair. The most general unifier, if it exists, is the coequaliser, in the full subcategory spanned by coproducts of representable presheaves. The main purpose of the pattern unification algorithm consists in constructing this coequaliser, if it exists, which is the case as long as there exists a unifier.

Related work

First-order unification was categorically described in [2]. Pattern unification was introduced in [3], as a particular case of higher-order unification for the simply-typed lambda-calculus, where metavariables are applied to distinct variables. It was categorically rephrased by [4], with concluding hints about how to generalise their work. The present work can be viewed as an explicit realisation of this generalisation.

Plan of the paper

In Section 3, we state our main result that justifies pattern unification algorithms. Then we tackle the proof algorithm, starting with the pruning phase (Section 5), the unification phase (Section 4), the occur-check (Section 6), and finally we justify completeness (Section 7).

General notations

If $\mathscr{B}$ is a category and a and b are two objects, we denote the set of morphisms between a and b by $\hom_{\mathscr{B}}(a, b)$ or $\mathscr{B}(a, b)$.

We denote the identity morphism at an object x by 1_x. We denote by $()$ any initial morphism and by ! any terminal morphism.

We denote the coproduct of two objects A and B by $A+B$ and the coproduct of a family of objects $(A_i)_{i\in I}$ by $\coprod_{i\in I} A_i$, and similarly for morphisms.

If $(g_i : A_i \to B)_{i\in I}$ is a family of arrows, we denote by $[g_i] : \coprod_{i\in I} A_i \to B$ the induced coproduct pairing. If $f : A \to B$ and $g : A' \to B$, we sometimes denote the induced morphism $[f, g] : A + A' \to B$ by merely f, g. Conversely, if $g : \coprod_{i\in I} A_i \to B$, we denote by g_i the morphism $A_i \to \coprod_i A_i \to B$

Coproduct injections $A_i \to \coprod_{i\in I} A_i$ are typically denoted by in_i.

Given an adjunction $L \dashv R$ and a morphism $f : A \to RB$, we denote by $f^* : LA \to B$ its transpose, and similarly, if $g : LA \to B$, then $g^* : A \to RB$. In particular, a Kleisli morphism $f : A \to TB$ induces a morphism $f^* : TA \to TB$ through the adjunction between the category of T-algebras and $\mathscr{C}$.

We denote the Kleisli composition of $f : A \to TB$ and $g : TB \to T\Gamma$ by $f[g] = g^* \circ f$. The unit of the monad T is denoted by η.

The left adjoint $\mathcal{L}$ to the Kleisli category of T postcomposes any morphism $A \to B$ by η.

2 General setting

2.1 Base category

We work in a full subcategory $\mathcal{C}$ of functors $\mathcal{A} \to \mathrm{Set}$, namely, those preserving finite connected limits, where $\mathcal{A}$ is a small category in which all morphisms are monomorphisms and has finite connected limits.

Example 1. For the category of nominal sets, take $\mathcal{A} = \mathbb{F}_m$ the category of finite cardinals and injections.

Remark 1. The restriction to functors preserving finite connected limits will justify that the case where we unify two metavariables: the result is then a new metavariable, whose arity is computed in $\mathcal{A}$.

The yoneda embedding $\mathcal{A}^{op} \to [\mathcal{A}, \mathrm{Set}]$ factors through $\mathcal{C} \to [\mathcal{A}, \mathrm{Set}]$. We denote the fully faithful embedding as $\mathscr{D} \xrightarrow{K} \mathcal{C}$. A useful lemma that we will exploit is the following:

Lemma 1. *$\mathcal{C}$ is closed under limits, coproducts, and filtered colimits.*

In this rest of this section, we abstract this situation by listing a number of properties that we will use in the following to describe the main unification phase.

Property 1. $K : \mathscr{D} \to \mathscr{C}$ is fully faithful.

Property 2. $\mathscr{C}$ is cocomplete..

Remark 2. We need those cocompleteness properties so that we can compute free monads of a finitary endofunctor as the colimit of an initial chain.

Notation 21. *We denote by $\mathscr{D}^+ \xrightarrow{K^+} \mathscr{C}$ the full subcategory of $\mathscr{C}$ consisting of finite coproducts of objects of $\mathscr{D}$.*

Notation 22. *A context $M : a_M, N : a_N, \ldots$ denotes the object $\coprod_{i\in\{M,N,\ldots\}} Ka_i \in \mathscr{D}^+$.*

We will be interested in coequalisers in the Kleisli category restricted to $\mathscr{D}^+$.

Property 3. $\mathscr{D}$ has finite connected colimits and K preserves them.

Property 4. Given any morphism $f : a \to b$ in $\mathscr{D}$, the morphism Kf is epimorphic.

Property 5. Coproduct injections $A_i \to \coprod_j A_j$ in $\mathscr{C}$ are monomorphisms.

The following two properties are direct consequences of Lemma 1.

Property 6. For each $d \in \mathscr{D}$, the object Kd is connected, i.e., any morphism $Kd \to \coprod_i A_i$ factors through exactly one coproduct injection $A_j \to \coprod_i A_i$.

2.2 The endofunctor for syntax

We assume given an endofunctor F on $[\mathcal{A}, \mathrm{Set}]$ such that $F(X)$ is of the shape $\coprod_{o \in O} \prod_{j \in J_o} X \circ L_{o,j} \times S_o$, where

- S_o is a functor preserving finite connected limits.
- J_o is a finite set
- $L_{o,j}$ is an endofunctor on $\mathcal{A}$ preserving finite connected limits.

Remark 3. S_o is, for instance, the variable presheaf (in this case, J_o is empty). The fact that it preserves pullbacks is crucial when unifying a metavariable with a variable, to show that either there is no unifier, either there is a coequaliser.

Example 2. Consider the endofunctor on Nom corresponding to λ-calculus: $F(X) = (1\times)I + X \times X + X \circ (- + 1)$, where I is the representable presheaf $y1$.

We will rely on the following result.

Lemma 2. *F is finitary and restricts as an endofunctor on the subcategory $\mathscr{C}$ of functors preserving finite connected limits.*

We now abstract this situation by stating the properties that we will need.

Notation 23. *Given an index $o \in O$, and $a \in \mathscr{D}$, we denote $\coprod_{j \in J_o} KL_{o,j}a$ by a^o. Given $Kf : Ka \to Kb$, we denote the induced morphism $a^o \to b^o$ by f^o.*

Property 7. A morphism $Ka \to FX$ is equivalently given by an index $o \in O$, a morphism $s : Ka \to S_o$, and a morphism $f : a^o \to X$.

Property 8. The endofunctor $F : \mathscr{C} \to \mathscr{C}$ is finitary.

Together with cocompleteness properties of $\mathscr{C}$ (Property 2), this ensures the existence of the (algebraically) free monad generated by F.

Notation 24. *We denote the free monad F^* on F by T.*

Notation 25. *Given an index $o \in O$, a morphism $s : Ka \to S_o$, and $f : a^o \to TX$, we denote the induced morphism*

$$Ka \to FTX \hookrightarrow TX$$

by $o(f; s)$, where the first morphism $Ka \to FTX$ is induced by Property 7.

Let $\Gamma = M_1 : a_1, \ldots, M_n : a_n \in \mathscr{D}^+$, following Notation 22.

Given $f \in \hom_{\mathscr{D}}(a, a_i)$, we denote the morphism

$$Ka \to Ka_i \hookrightarrow \Gamma \xrightarrow{\eta} T(\Gamma)$$

by $M_i(f)$.

Property 9. Let $\Gamma = M_1 : a_1, \ldots, M_n : a_n \in \mathscr{D}^+$. Then, any morphism $u : Ka \to T\Gamma$ is one of the two mutually exclusive following possibilities:

– $M_i(f)$ for some unique i and $f : a \to a_i$,
– $o(f; s)$ for some unique $o \in O$, $f : a^o \to T\Gamma$ and $s : Ka \to S_o$.

We say that u is *flexible (flex)* in the first case and *rigid* in the other case.

Lemma 3. *Let $\Gamma = M_1 : a_1, \ldots, M_n : a_n \in \mathscr{D}^+$ and $g : \Gamma \to T\Delta$. Then, for any $o \in O$, $f : a^o \to T\Gamma$ and $s : Ka \to S_o$, we have $o(f; s)[g] = o(f[g]; s)$, and for any $1 \leq i \leq n$, $x : b \to a_i$, we have $M_i(x)[g] = g_i \circ Kx$.*

Moreover, for any $u : Kb \to Ka$,

$$o(f; s) \circ Ku = o(f \circ u^o; s \circ Ku)$$

We end this section by introducing notations for Kleisli morphisms.

Notation 26. *Let Γ and Δ be contexts and $a \in \mathscr{D}$. Any $t : Ka \to T(\Gamma + \Delta)$ induces a Kleisli morphism $\Gamma, M : a \to T(\Gamma + \Delta)$ that we denote by $M \mapsto t$.*

3 Main result

The main point of pattern unification is that a coequaliser diagram in Kl_T selecting objects in $\mathscr{D}^+$ either has no unifier, either has a colimiting cocone.

Because working this logical disjunction is slightly inconvenient, we rephrase it as a true coequaliser by freely adding a terminal object.

Definition 1. *Given a category $\mathscr{B}$, let $\mathscr{B}^*$ be $\mathscr{B}$ extended freely with a terminal object.*

Adding a terminal object results in adding a terminal cocone to all diagrams. As a consequence, we have the following lemma.

Lemma 4. *Let J be a diagram in a category $\mathscr{B}$. The following are equivalent:*

1. *J has a colimit has long as its category of cocones is not empty.*
2. *J has a colimit in $\mathscr{B}^*$.*

Thus, we are going to work in Kl_T^* rather than Kl_T.
The following result is also useful.

Lemma 5. *Given a category $\mathscr{B}$, the canonical functor $\mathscr{B} \to \mathscr{B}^*$ creates colimits.*

This has the following useful consequences:

1. whenever the colimit in Kl_T^* is not the terminal object, it is also a colimit in Kl_T;
2. existing colimits in Kl_T are also colimits in Kl_T^*;
3. in particular, coproducts in Kl_T (which are computed in $\mathscr{C}$) are also coproducts in Kl_T^*.

Notation 31. *We denote by $\top$ the terminal object and by $!$ any terminal morphism.*

Here is our main result.

Theorem 1. *Let $Kl^*_{\mathscr{D}^+}$ be the full subcategory of Kl^*_T consisting of objects of $\mathscr{D}^+ \cup \{\top\}$. Then, $Kl^*_{\mathscr{D}^+}$ has coequalisers. Moreover, the inclusion $Kl^*_{\mathscr{D}^+} \to Kl^*_T$ preserves them. We need to say somewhere that this Kleisli category is the same as the Kleisli category of the monad on $[\mathcal{A}, \mathrm{Set}]$, restricted to $\mathscr{C}$ or $\mathscr{D}^+$.*

In other words, any coequaliser diagram $A \rightrightarrows TB$ in Kl_T where A and B are in $\mathscr{D}^+$ has a colimit $B \to T\Gamma$ where $C \in \mathscr{D}^+ \cup \{\top\}$: either such a coequaliser has a colimit (which can be computed in $Kl_{\mathscr{D}^+}$), either there is no cocone at all.

4 Unification phase

In this section, we describe the main unification phase, which relies on the pruning phase, as we shall see. The goal is to compute a coequaliser in Kl^*_T, where the domain is in $\mathscr{D}^+$, as in

$$\begin{tikzcd} \coprod_i Ka_i \arrow[r, shift left, "t"] \arrow[r, shift right, "u"'] & \Gamma \arrow[r, dotted, "\sigma"] & \Delta \end{tikzcd}$$

We denote such a situation by

$$\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$$

The intuition behind this notation is that the symbol $\Rightarrow$ separates the input and the output of the algorithm, that we are going to describe with inductive rules.

Let us start with simple cases:

- when $\Gamma = \top$. Then, the pushout is the terminal cocone:

$$\frac{}{\top \vdash t = u \Rightarrow\ ! \dashv \top};$$

- when the coproduct is empty, the coequaliser is just Γ:

$$\frac{}{\Gamma \vdash () = () \Rightarrow 1_\Gamma \dashv \Gamma}.$$

Furthermore, when the coproduct is neither empty nor a singleton, the coequaliser can be decomposed thanks to the following rule:

$$\frac{\Gamma \vdash t_1 = u_1 \Rightarrow \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash t_2[\sigma_1] = u_2[\sigma_1] \Rightarrow \sigma_2 \dashv \Delta_2}{\Gamma \vdash t_1, t_2 = u_1, u_2 \Rightarrow \sigma_1[\sigma_2] \dashv \Delta_2} \tag{1}$$

This is justified by a general stepwise construction of coequalisers:

Lemma 6. *Let $f_1, g_1 : A_1 \to B$ and $f_2, g_2 : A_2 \to B$ be morphisms in some category.*

If the following two diagrams as coequalisers

$$\begin{tikzcd}
A_1 \arrow[r, bend left, "f_1"] \arrow[r, bend right, "f_1'"'] & B \arrow[r, "u"] & C
\end{tikzcd}
\qquad
\begin{tikzcd}
 & B \arrow[dr, "u"] & & \\
A_2 \arrow[ur, "f_2"] \arrow[dr, "f_2'"'] & & C \arrow[r, "v"] & D \\
 & B \arrow[ur, "u"'] & &
\end{tikzcd}$$

Then this one as well:

$$\begin{tikzcd}
A_1 + A_2 \arrow[r, bend left, "{f_1,f_2}"] \arrow[r, bend right, "{f_1',f_2'}"'] & B \arrow[r, "v \circ u"] & D
\end{tikzcd}$$

What remains to be addressed is the case where the coproduct is a singleton and $\Gamma = \coprod_j Kb_j$, that is, a coequaliser diagram

$$\begin{tikzcd}
Ka \arrow[r, shift left, "t"] \arrow[r, shift right, "u"'] & T\Gamma
\end{tikzcd}$$

By Lemma 9, $t, u : Ka \to T\Gamma$ are one of the two following possibilites:

- $M(f)$ for some $f : a \to b_j$
- $o(f; s)$ for some $o \in O$, $f : a^o \to T\Gamma$ and $s : Ka \to S_o$.

In the next subsections, we discuss the different possibilities.

4.1 Rigid-rigid

Here we are in the situation where the parallel morphisms $t, u : Ka \to T\Gamma$ are $o(f; s)$ and $o'(f'; s')$ for some $o, o' \in O$, morphisms $f : a^o \to T\Gamma$, $f' : a^{o'} \to T\Gamma$, and morphisms $s : Ka \to S_o$ and $s' : Ka \to S_{o'}$.

Assume given a unifier, i.e., a Kleisli morphism $\sigma : \Gamma \to T\Delta$ such that $t[\sigma] = u[\sigma]$. By Lemma 3, this means that $o(f[\sigma]; s) = o'(f'[\sigma]; s')$. By Lemma 9, this implies that $o = o'$, $f[\sigma] = f'[\sigma]$, and $s = s'$.

Therefore, we get the following error rules

$$\frac{o \neq o'}{\Gamma \vdash o(f; s) = o'(f'; s') \Rightarrow\ ! \dashv \top} \qquad \frac{s \neq s'}{\Gamma \vdash o(f; s) = o(f'; s') \Rightarrow\ ! \dashv \top}$$

We now assume $o = o'$ and $s = s'$. Then, σ unifies t and u if and only if it unifies f and f'. Therefore, we get the following rule

$$\frac{\Gamma \vdash f = g \Rightarrow \sigma \dashv \Delta}{\Gamma \vdash o(f; s) = o(g; s) \Rightarrow \sigma \dashv \Delta} \tag{2}$$

4.2 Flex-*

In this section, we consider a coequaliser diagram $Ka \underset{u}{\overset{t}{\rightrightarrows}} T\Gamma$ such that $\Gamma = \Gamma', M : b$ and $t = M(f)$ for some $f : a \to b$. We need to distinguish three cases:

1. u is $M(g)$ for some g.
2. M does not occur in u, in the sense that $u : Ka \to T\Gamma$ factors through $T\Gamma' \xrightarrow{Tin_{\Gamma'}} T\Gamma$.
3. u is $o(g; s)$ and M does occur in $g : a^o \to T\Gamma$, in the sense that g does not factor through $T\Gamma' \xrightarrow{Tin_{\Gamma'}} T\Gamma$.

In Section 6, we show that in the third case there is no unifier, justifying the rule

$$\frac{T\Gamma' \xrightarrow{Tin_{\Gamma'}} T\Gamma \text{ does not factor } g : a^o \to T\Gamma}{\Gamma', M : b \vdash o(g; s) :> f \Rightarrow !; ! \dashv \top}$$

We investigate the other cases below.

Flex-Flex Here we want to unify $M(f)$ and $M(g)$, with $f, g \in \hom_{\mathscr{D}}(a, b)$, that is, we want to coequalise, in Kl_T^*, the following morphisms

$$\begin{tikzcd}
 & Kb \arrow[r, "\eta"] & TKb \arrow[dr, "Tin_M"] & \\
Ka \arrow[ur, "Kf"] \arrow[dr, "Kg"'] & & & T(\dots, M : b) \\
 & Kb \arrow[r, "\eta"'] & TKb \arrow[ur, "Tin_M"'] &
\end{tikzcd}$$

Note that this is coqualiser diagram of the shape (in Kl_T^*) of the shape

$$\begin{tikzcd}
 & Kb \arrow[dr] & \\
Ka \arrow[ur, "\mathcal{L}Kf"] \arrow[dr, "\mathcal{L}Kg"'] & & Kb + \dots \\
 & Kb \arrow[ur] &
\end{tikzcd} ,$$

where $\mathcal{L}$ denotes the left adjoint $\mathscr{C} \to Kl_T$, postcomposing morphisms with η.

Therefore, the category of cocones is isomorphic to the category of cocones coequalising $\mathcal{L}Kf$ and $\mathcal{L}Kg$. Since $\mathcal{L}$ is left adjoint (and thus preserves coequalisers) and K preserves coequalisers (Property 3), the initial of f and g in $\mathcal{D}$. Therefore, we get the rule

$$\frac{Ka \underset{Kg}{\overset{Kf}{\rightrightarrows}} Kb \overset{Kh}{\dashrightarrow} Kc \text{ is a coequaliser}}{\Gamma, M : b \vdash M(f) = M(g) \Rightarrow M \mapsto M'(h) \dashv \Gamma, M' : c}$$

And since $\mathscr{D}$ has finite connected colimits (Property 3), given any input, there is always an output such that the rule applies.

4.3 Flex-rigid (successful occur-check)

In this section, we consider a coequaliser diagram $\begin{tikzcd} Ka \arrow[r, shift left, "t"] \arrow[r, shift right, "u"'] & T\Gamma \end{tikzcd}$ such that $\Gamma = \Gamma', M : b$ and $t = M(f)$ for some $f : a \to b$, and u factors as $Ka \xrightarrow{g} T\Gamma' \hookrightarrow T\Gamma$.

Thus, the coequaliser

$$\begin{tikzcd} Ka \arrow[r, shift left, "t"] \arrow[r, shift right, "u"'] & T\Gamma \end{tikzcd}$$

is a coequaliser (in Kl^*_T) of the shape

$$\begin{tikzcd} & Kb \arrow[dr] & \\ Ka \arrow[ur, "\mathcal{L}f"] \arrow[dr, "g"'] & & \Gamma', M : b \\ & \Gamma' \arrow[ur] & \end{tikzcd} , \tag{3}$$

Lemma 7. *Consider un pushout diagram in a category*

$$\begin{tikzcd} A \arrow[r] \arrow[d] & B \arrow[d, "u"] \\ C \arrow[r, "v"'] & D \end{tikzcd}$$

Then, the following diagram is a coequaliser

$$\begin{tikzcd} & B \arrow[dr, "in_1"] & & \\ A \arrow[ur] \arrow[dr] & & B + C \arrow[r, "{[u,v]}"'] & D \\ & C \arrow[ur, "in_2"'] & & \end{tikzcd} ,$$

Therefore, computing the coequaliser of (3) amounts to computing a pushout in $Kl^*_{\mathscr{D}+}$.

$$\begin{tikzcd} Ka \arrow[r, "\mathcal{L}f"] \arrow[d, "g"'] & Kb \arrow[d, dotted, "v"] \\ \Gamma \arrow[r, dotted, "\sigma"'] & \Delta \end{tikzcd} ,$$

As we shall see in Section (5), this is expressed by the statement $\Gamma \vdash g :> f \Rightarrow v; \sigma \dashv \Delta$. Therefore, we have the rule

$$\frac{\Gamma \vdash g :> f \Rightarrow v; \sigma \dashv \Delta}{\Gamma, M : b \vdash M(f) = g \Rightarrow \sigma, M \mapsto v \dashv \Delta}$$

5 Pruning phase

The pruning phase corresponds to computing a pushout diagram in Kl^*_T where one branch is a finite coproduct of free morphisms.

Consider a pushout in $Kl^*_{\mathscr{D}+}$

$$\begin{tikzcd}
\coprod_i Ka_i \arrow[r, "\coprod_i \mathcal{L}Kf_i = \mathcal{L}\coprod_i KF_i"] \arrow[d, "g"'] & \coprod_i Kb_i \arrow[d, dotted, "u"] \\
\Gamma \arrow[r, dotted, "\sigma"'] & \Delta \arrow[ul, phantom, very near start, "\ulcorner"]
\end{tikzcd}$$

We denote such a situation by

$$\Gamma \vdash g :> \coprod_i Kf_i \Rightarrow u; \sigma \dashv \Delta$$

Remark 4. For the intuition behind the notation and the relation to the so-called pruning process, consider the case of λ-calculus, in the category Nom of nominal sets. A span $\Gamma \xleftarrow{g} Kn \xrightarrow{\mathcal{L}f} Km$ corresponds to a term in $t \in T\Gamma_n$ and a choice of distinct m variables in $\{0, \ldots, n-1\}$, that is, an injection $f : m \to n$. Then, the pushout, if it exists, consists in "coercing" (hence the symbol :>) the term t to live in $T\Gamma_m$, by restricting the arity of the metavariables, as $\sigma : \Gamma \to T\Delta$ does. The resulting term $u \in \hom(Kn, T\Delta) \cong T\Delta_n$ is t but living in the "smaller" context $\{0, \ldots, m-1\}$ and with the restricted metavariables.

Remark 5. A cocone consists in morphisms $\coprod_i Kb_i \xrightarrow{t} T\Delta \xleftarrow{\sigma} \Gamma$ such that $g[\sigma] = t \circ f^o$, i.e., for all $i \in I$, we have $g_i[\sigma] = t_i \circ Kf$. Note that given σ, there is at most one t that works because Kf is epimorphic.

The simplest case is when the coproduct is empty: then, the pushout is Γ.

$$\frac{}{\Gamma \vdash () :> () \Rightarrow (); 1_\Gamma \dashv \Gamma}$$

Another simple case is when $\Gamma = \top$. Then, the pushout is the terminal cocone. Thus we have the rule

$$\frac{}{\top \vdash\ ! :> f \Rightarrow\ !; ! \dashv \top}$$

The pushout can be decomposed into smaller components, thanks to the following lemma.

Lemma 8. *In any category, if the following two diagrams are pushouts,*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & A' \arrow[d, dashed, "u"] \\
X \arrow[r, dashed, "\sigma"'] & Z
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, "f'"] \arrow[d, "g'"'] & B' \arrow[dd, dashed, "u'"] \\
X \arrow[d, "\sigma"'] & \\
Z \arrow[r, dashed, "\sigma'"'] & Z'
\end{tikzcd}$$

then this one as well:

$$\begin{tikzcd}
A+B \arrow[r, "f+f'"] \arrow[d, "{[g,g']}"'] & A'+B' \arrow[d, dashed, "{[\sigma'\circ u,u']}"] \\
X \arrow[r, dashed, "\sigma'\circ\sigma"'] & Z'
\end{tikzcd}$$

Thus we have the following rule.

$$\frac{\Gamma \vdash g_1 :> f_1 \Rightarrow u; \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash g_2 :> f_2[\sigma_1] \Rightarrow u_2; \sigma_2 \dashv \Delta_2}{\Gamma \vdash g_1, g_2 :> f_1 + f_2 \Rightarrow u_1[\sigma_2], u_2; \sigma_1[\sigma_2] \dashv \Delta_2} \tag{4}$$

Thanks to the previous rule, we can focus on the case where the coproduct is the singleton (since we focus on finite coproducts of elements of $\mathscr{D}$). Thus, we want to compute the pushout

$$\begin{tikzcd}
Ka \arrow[r, "Kf"] \arrow[d, "u"'] & Kb \arrow[r, "\eta"] & TKb \\
T\Gamma & &
\end{tikzcd}$$

By Lemma 9, the left vertical morphism $u : Ka \to T\Gamma$ is one of the two following possibilites:

- $M(g)$ for some $g : a \to c$, where $\Gamma = \Gamma', M : c$
- $o(g; s)$ for some $o \in O$, $g : a^o \to T\Gamma$, and $s : Ka \to S_o$.

5.1 Flex

Here, we are in the situation where we want to compute the pushout of free morphisms in Kl_T

$$\begin{tikzcd}
Ka \arrow[r, "\mathcal{L}Kf"] \arrow[d, "\mathcal{L}Kg"'] & Kb \\
Kc \arrow[d, "in_M"'] & \\
\Gamma, M : c &
\end{tikzcd}$$

Thanks to the following lemma, it is enough to compute the pushout of $\mathcal{L}Kf$ and $\mathcal{L}Kg$.

Lemma 9. *In any category, if the following diagram is a pushout*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, dashed, "u"] \\
X \arrow[r, dashed, "\sigma"'] & Z
\end{tikzcd}$$

then this one as well:

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, dashed, "u"] \\
X \arrow[d, "in_1"'] & Z \arrow[d, dashed, "in_1"] \\
X + Y \arrow[r, dashed, "\sigma+Y"'] & Z + Y
\end{tikzcd}$$

Since $\mathcal{L}$ is left adjoint (and thus preserves coequalisers) and K preserves pushouts (Property 3), the pushout can be computed in $\mathscr{D}$. Therefore, we get the rule

$$\frac{\begin{tikzcd}
Ka \arrow[r, "Kf"] \arrow[d, "Kg"'] & Kb \arrow[d, dotted, "Kg'"] \\
Kc \arrow[r, dotted, "Kf'"'] & Kd
\end{tikzcd} \quad \text{is a pushout}}{\Gamma, M : c \vdash M(g) :> f \Rightarrow M'(g'); M \mapsto M'(f') \dashv \Gamma, M' : d}$$

5.2 Rigid

We want to compute the pushout

$$\begin{tikzcd}
Ka \arrow[r, "Kf"] \arrow[d, "o(g;s)"'] & Kb \arrow[r, "\eta"] & TKb \\
T\Gamma & &
\end{tikzcd}$$

where $g : a^o \to T\Gamma$ and $s : Ka \to S_o$.

By Remark 5, a cocone in Kl_T is given by an object Δ with morphisms $Kb \overset{u}{\to} T\Delta \overset{\sigma}{\leftarrow} \Gamma$ such that $o(g; s)[\sigma] = u \circ Kf$. By Lemma 3, this means that $o(g[\sigma]; s) = u \circ Kf$. Now, u is either some $M'(f')$ or $o'(g'; s')$, with $g' : b^o \to T\Delta$ and $s' : Kb \to S_{o'}$. But in the first case, $u \circ Kf = M'(f') \circ Kf = M'(f' \circ f)$ so it cannot equal $o(g[\sigma]; s)$. So we are in the second case, and for the same reason, $o = o'$, $g[\sigma] = g' \circ f^o$ and $s = s' \circ Kf$. Note that such a s' is unique because Kf is epimorphic. Therefore we get the rule

$$\frac{Kf : Ka \to Kb \text{ does not factor } s : Ka \to S_o}{\Gamma \vdash o(g; s) :> f \Rightarrow !; ! \dashv \top} \tag{5}$$

Remark 6. If S_o is orthogonal [1, Definition 1.32] to all morphisms (i.e., given any span $S_o \leftarrow a \to b$, there exists a unique $b \to S_o$ completing the triangle), as in the case where S_o is the output type "dirac", then this rule never applies.

$$\frac{\Gamma \vdash g :> f^o \Rightarrow u; \sigma \dashv \Delta \qquad s = s' \circ Kf}{\Gamma \vdash o(g; s) :> f \Rightarrow o(u; s'); \sigma \dashv \Delta} \tag{6}$$

6 Occur-check

The occur-check allows to jump from the main unification phase (Section 4) to the pruning phase (Section 5), whenever the metavariable appearing at the toplevel of the l.h.s does not appear in the r.h.s. This section is devoted to the proof that if there is a unifier, then the metavariable does not appear on the r.h.s, either it appears at top-level (see Corollary 1).

We prove it by working in the category $[\mathcal{A}, \mathrm{Set}]$ of presheaves over $\mathscr{D}$. We denote by M the monad on $[\mathcal{A}, \mathrm{Set}]$ that restricts to the monad T on $\mathscr{C}$.

6.1 Preliminary lemma

There is a M-algebra structure $M\mathbb{N} \to \mathbb{N}$ on the constant presheaf mapping any object to the set $\mathbb{N}$ of natural numbers that adds one to the maximum of the arguments. For any $s : X \to \mathbb{N}$, this induces an algebra morphism $MX \xrightarrow{|-|_s} \mathbb{N}$, which basically computes the maximal depth of an element of MX, with custom depth for leaves in X as specified by s.

Definition 2. *Let $s : X \to \mathbb{N}$. A natural transformation $n : A \to MX$ is said* s-flat *if $A \to MX \xrightarrow{|-|_s} \mathbb{N}$ is constant. It is said* flat *if there exists s that makes it s-flat.*

In practice we will have X as the initial presheaf, or the terminal presheaf.

Example 3. Any natural transformation out of a representable presheaf is 0-flat.

The crucial lemma is the following.

Proposition 1. *Let $A \xrightarrow{n} M\emptyset$ be a flat morphism. Then, the following diagram is a pullback*

$$\begin{tikzcd}
A + A \times_{M\emptyset} A \arrow[r] \arrow[d] & MA \arrow[d, "n^*"] \\
A \arrow[r, "n"'] & M\emptyset
\end{tikzcd}$$

As an example, if $\mathscr{D}$ is discrete, and if n is $yd \to M\emptyset$, choosing an element $u \in M\emptyset_d$, then what it says is that given any element $t \in MA_d$ such that $t[u] = u$, either $t = \eta(id_d)$, i.e., t is a metavariable, either $t = u$.

Corollary 1. *Given $\sigma : \Gamma \to T1$, $u : Kd \to T1$, $f : Kc \to Kd$, if the following square commutes, then the top morphism factors through either $T\Gamma \to T(Kd+\Gamma)$ or $Kd \to TKd$*

$$\begin{tikzcd} Kc \arrow[r] \arrow[d, "f"'] & T(Kd+\Gamma) \arrow[d, "{[u,\sigma]^*}"] \\ Kd \arrow[r, "u"'] & T1 \end{tikzcd}$$

This concludes the argument, since any unifier induces a unifier targeting $T1$, by postcomposing with $T!$.

7 Completeness

Each inductive rule presented so far provides an elementary step for the construction of coequalisers. Since the rules were already justified to be sound, the following two properties are sufficient to ensure that applying rules eagerly eventually leads to a coequaliser:

- progress, i.e., there is always one rule that applies (Section (7.1));
- termination, i.e., there is no infinite sequence of rule applications (Section (7.2)).

Before delving into the details, we first need to add a side condition to the stepwise rules (1) and (4), so that termination is not obviously wrong: we enforce that the coproduct is split in two non-empty components.

7.1 Progress

Progress for the main unification phase means that given a coequalising diagram $\coprod_i Ka \overset{t}{\underset{u}{\rightrightarrows}} \Gamma$, there is always a step that we can perform, i.e., there is always one rule that applies, in some sense that can be made precise for each rule, roughly meaning Γ, t, and u are of the shape Γ', t' and u' (up to isomorphism) when the conclusion of the rule is $\Gamma' \vdash t' = u' \Rightarrow \ldots$, and all the side conditions in the premises are satisfied (as, for example, the only premise of (5)). Similar definitions are required for the pruning phase.

Note that if we do not add the above mentioned side conditions to the stepwise rules (1) and (4), they always apply.

Proposition 2. *Given an input* $\coprod_i Ka \overset{t}{\underset{u}{\rightrightarrows}} \Gamma$ *, there is always a rule of the main unification phase that applies. Given an input*

$$\begin{tikzcd}
\coprod_i Ka_i \arrow[r, "\coprod_i Kf_i"] \arrow[d, "g"'] & \coprod_i Kb_i \arrow[r, "\eta"] & T\coprod_i Kb_i \\
X & &
\end{tikzcd}$$

there is always one rule in the pruning phase that applies.

7.2 Termination

The usual termination argument for pattern unification applies. Roughly, it consists in defining the size of an input and realising that it strictly decreases in the premises. This relies on the notion of the size $|\Gamma|$ of a context Γ (as an element of $\mathscr{D}^+$), which can be defined without ambiguity as the size of the coproducts, thanks to the following lemma, because objects of $\mathscr{D}$ are connected (Property 6).

Lemma 10. *Let* $\coprod_{i\in I} A_i$ *and* $\coprod_{j\in J} B_j$ *be isomorphic coproducts of connected objects. Then,* I *and* J *are isomorphic sets.*

We extend the definition of $|\Gamma|$ to the case where $\Gamma = \top$, by taking $|\top| = 0$.

We also need to define the size $||t||$ of a *term* $t : Kd \to \Delta$ as $|t|_0$, as defined in Section 4.3.

First, we sketch termination of the pruning phase 5, and then termination of the main unification phase 4.

Pruning phase There are two recursives rules: (4) and (6).

In fact, they can be straightforwardly recast so that the input f is structurally decreasing in $\Gamma \vdash f :> g \Rightarrow u; \sigma \dashv \Delta$.

Main unification phase There are two recursives rules: (1) and (2). Contrary to the pruning phase, the first one is not structurally recursive since a substitution is applied to the argument.

Lemma 11. *If there is a finite derivation tree of* $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$*, then*

- $|\Gamma| \geq |\Delta|$
- *if* $|\Gamma| = |\Delta|$*, then* σ *is a renaming, in the sense that it instantiates metavariables with metavariables.*

This relies on the following lemma about the pruning phase, easily proved by induction.

Lemma 12. *If there is a finite derivation tree of*

$$\Gamma \vdash f :> g \Rightarrow u; \sigma \dashv \Delta$$

and $\Delta \neq \top$, , then $|\Gamma| = |\Delta|$ and σ is a renaming, *i.e., a free morphism: it factors through $\Delta \to T\Delta$.*

Next, the crucial point, in the stepwise rule (1), is the following lemma.

Lemma 13. *If σ is a renaming, then $||t[\sigma]|| = ||t||$.*

Conclusion and future work

It would be nice to allow metavariables with mixed linearity constraints.

References

1. J. Adámek and J. Rosicky. *Locally Presentable and Accessible Categories.* Cambridge University Press, 1994.
2. 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.
3. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.
4. Andrea Vezzosi and Andreas Abel. A categorical perspective on pattern unification. *RISC-Linz*, page 69, 2014.