General Pattern unification

August 2, 2022

We give a general correctness proof for pattern unification in a categorical setting. Beyond simply typed syntax with binders, our categorical proof handles unification for linear or (intrinsic) polymorphic syntax such as system F (Section 10). In Section 3, we propose a general categorical setting in which pattern unification applies. In this case, a second-order syntax with metavariables applied to distinct variables (according to the pattern restriction) corresponds to a free monad applied a coproduct of representable presheaves.

In Section 4, we state our main result that justifies pattern unification algorithms. Then we tackle the proof algorithm, starting with the pruning phase (Section 6), the coequalising phase (Section 7), the occur-check phase (Section 8), and finally we justify termination (Section 9).

Some benefits: we unify the operation and the variable case.

1 Introduction

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

1.1 A short introduction to pattern unification

In a unification, we are interested 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 factors uniquely any other unifier.

Pattern unification consists in restricting the arguments of a metavariable 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 $M(x_1, \dots, x_n)$ at the top level. It then enters a so-called *occur-check* phase where it checks whether this metavariable occurs in the other handside u. If not, it enters a so-called *pruning phase* where it tries to recursively remove all occurences of variables in u that are not among $x_1, \dots, x_n$, by restricting the arity of the occuring metavariables. If the occur-check is negative, 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 Generalising 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 Set, the category of sets. A functor $X : \mathbb{F}_m \to \mathrm{Set}$ can be thought of as assigning to each cardinal n a set X_n of expressions with n free variables. The action on morphisms of $\mathbb{F}_m$ means that these expressions come equipped with injective renamings. The λ-calculus, for instance, is a functor Λ satisfying the recursive equation $\Lambda_n \cong n+\Lambda_n\times\Lambda_n+\Lambda_{n+1}$, where n denotes the n^{th} cardinal (for the free variables), and $- + -$ is disjoint union.

In pattern unification, we consider extensions 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 to the list of arguments of the metavariable. In other words, $Inj(p, n)$ is just the homset $\mathbb{F}_m(p, n)$.

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, 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 $\Lambda' \cong \Sigma(\Lambda') + yp$. Another way to put it is that Λ' is the free algebra on yp. Therefore, denoting T the free algebra monad, Λ is just $T(0)$ and Λ' is $T(yp)$. Now, if we want to extend the syntax with another metavariable of arity q, then the syntax is obtained as $T(yp + yq)$.

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, together with an endofunctor Σ on it. Then, the abstract version of a syntax extended with metavariables is T applied to a finite coproduct of representable presheaves.

Now, we state the unification problem in this abstract setting. To this end, let us look at 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, in the case of λ-calculus, 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 $t[i \mapsto x_i]$. 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

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

corresponding to selecting a pair of terms with p free variables and involving metavariables of arity $q_1, \ldots, q_n$. 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.

2 Related work

First-order unification was categorically described in [2].

[5] introduces pattern unification, a particular case of higher-order unification for the simply-typed lambda-calculus, where metavariables are applied to distinct variables.

[6] provides a categorical understanding of pattern unification in this setting, and gives some hints on how to generalise their account. In this work, we follow this path and provide an explicit realisation of this generalisation.

3 General setting

3.1 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 regular 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 2. 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 $\mathcal{A}$.

Remark 3. Regularity of monomorphisms (i.e., the fact that monomorphisms are equalisers) implies their effectivity (i.e., they are the equalisers of their cokernel pairs) because the category has pullbacks. In fact, this condition is not strictly needed, but it will allow us (by Property 5 below) to convert a factorisation problem (does $g : A \to X$ factors through $e : A \twoheadrightarrow B$?) into an equality check (does e coequalises the kernel pair of g?), during the pruning phase (see Equation 2) making our description more effective. This happens, for instance, when checking that a variable occurs or not in some metavariable application.

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

Lemma 4. *Limits, coproducts, and filtered colimits in $\mathcal{C}$ are computed pointwise.*

Proof. All we have to check is that limits, coproducts, and filtered colimits of functors preserving finite connected limits still preserve finite connected limits. The case of limits is clear, since limits commute with limits. The case of coproducts follows from connected limits commuting with coproducts in Set. The case of filtered colimits follows from finite limits commuting with filtered colimits in Set. □

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 coequalising phase.

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

Property 2. *$\mathcal{C}$ is complete, has coproducts and filtered colimits.*

Proof. $\mathcal{C}$ is the category of models of a limit sketch, and thus is locally presentable, by [1, Proposition 1.51]. As a result, it is bicomplete [1, Remark 1.56]. □

Remark 5. We need cocompleteness so that we can compute free monads of a finitary endofunctor as the colimit of an initial chain. Completeness ensures that this free monad is algebraically free (not sure if it is helpful).

Notation 6. We denote by $\mathcal{D}^+ \xrightarrow{K^+} \mathcal{C}$ the full subcategory of $\mathcal{C}$ consisting of finite coproducts of objects of $\mathcal{D}$.

Definition 7. We will be interested in coequalisers in the Kleisli category restricted to $\mathcal{D}^+$.

Property 3. *$\mathcal{D}$ is closed under finite connected colimits.*

This is to deal with the case $M(\vec{x}) = N(\vec{y})$.

Proof. Because K is fully faithful, an equivalent statement is that $\mathcal{D}$ is closed under finite connected colimits and K preserves them. Now, we assumed that $\mathcal{A}$ has finite connected limits. Let us show that the yoneda embedding preserves them. We have a natural isomorphism $[\mathcal{A}, \mathrm{Set}](ya, JX) \simeq \mathcal{C}(Ka, X)$, where y is the yoneda embedding in $\mathcal{A}^o \to [\mathcal{A}, \mathrm{Set}]$, and $J : \mathcal{C} \to [\mathcal{A}, \mathrm{Set}]$ the canonical embedding. Now consider a finite connected limit $\lim F$ in $\mathcal{A}$. Then,

$$\begin{aligned}
\mathcal{C}(K\lim F, X) &\cong [\mathcal{A}, \mathrm{Set}](y\lim F, JX) \\
&\cong JX(\lim F) && \text{(By the Yoneda Lemma.)} \\
&\cong \lim(JX \circ F) && \text{(By definition of } \mathcal{C}\text{)} \\
&\cong \lim([\mathcal{A}, \mathrm{Set}](yF-, JX)] && \text{(By the Yoneda Lemma)} \\
&\cong \lim \mathcal{C}(KF-, X) \\
&\cong \mathcal{C}(\mathrm{colim} KF, X) && \text{(By left continuity of the hom-set bifunctor)}
\end{aligned}$$

Thus, $K\lim F \cong \mathrm{colim} KF$. □

Property 4. *Given any morphism $f : A \to B$ in $\mathcal{D}$, Kf is epimorphic.*

Proof. A morphism $f : A \to B$ is epimorphic if and only if the following square is a pushout

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "f"'] & B \arrow[d, equal] \\
B \arrow[r, equal] & B
\end{tikzcd}$$

We conclude by Propery 3, because all morphisms in $\mathcal{D} = \mathcal{A}^o$ are epimorphic by assumption. □

In fact, we have the following stronger property, that we state separately because as argued in Remark 3, this is not strictly required.

Property 5. *Given any morphism $f : A \to B$ in $\mathcal{D}$, the epimorphism Kf is effective, i.e, the following diagram is a coequaliser*

$$\begin{tikzcd}
\mathrm{Ker}(f) \arrow[r, bend left, "p_1"] \arrow[r, bend right, "p_2"'] & A \arrow[r, "f"] & B
\end{tikzcd}$$

where $\mathrm{Ker}(f) \rightrightarrows A$ *is the kernel pair of f defined as the pullback of f with itself, as in*

$$\begin{tikzcd}
\mathrm{Ker}(f) \arrow[r, "p_1"] \arrow[d, "p_2"'] & A \arrow[d, "f"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

of its kernel pair

Proof. Since $\mathcal{C}$ has pullbacks (Property 2), it is enough to show that Kf is regular. But since f is a regular epimorphism by assumption, we know that the f is the coequaliser of some $u_1, u_2 : C \to A$ in $\mathcal{D}$. We conclude by Property 3. □

Property 6. *Coproduct injections $A_i \to \coprod_j A_j$ in $\mathcal{C}$ are monomorphisms.*

Proof. This follows from Lemma 4, because a morphism $f : A \to B$ is monomorphic if and only if the following square is a pullback

$$\begin{tikzcd} A \arrow[r, equal] \arrow[d, equal] & A \arrow[d, "f"] \\ A \arrow[r, "f"'] & B \end{tikzcd}$$

□

The following two properties are direct consequences of Lemma 4.

Property 7. *For each $d \in \mathcal{D}$, the object Kd is connected, i.e., any morphism $Kd \to \coprod_i A_i$ factors through exactly one coproduct injection $A_j \to \coprod_i A_i$.*

Property 8. *For each $d \in \mathcal{D}$, the object Kd is finitely presentable, i.e., $\mathcal{C}(Kd, -)$ preserves filtered colimits.*

This is used to prove correctness of the occur-check by induction.

3.2 The endofunctor for syntax

We assume given an endofunctor F on $[\mathcal{A}, \mathrm{Set}]$ such that $F(X)$ is of the shape $\coprod_i \prod_{j \in J_i} X \circ L_{i,j} \times S_i$, where

- S_i is a functor preserving finite connected limits.
- J_i is a finite set
- $L_{i,j}$ is an endofunctor on $\mathcal{A}$.

Remark 8. S_i is, for instance, the variable object. The fact that it preserves pullbacks ensures that when unifying a metavariable with a variable, either there is no unifier, either there is a coequaliser.

Example 9. 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 10. *F is finitary and restricts as an endofunctor on the subcategory $\mathcal{C}$ of functors preserving finite connected limits.*

Proof. F is finitary because filtered colimits commute with finite limits and colimits. It restricts as stated because finite connected limits commute with coproducts and limits. □

Now, let us abstract this situation by stating the properties that we will need.

Property 9. *The endofunctor $F : \mathcal{C} \to \mathcal{C}$ is finitary.*

Proof. This is a straightforward consequence of Lemmas 10 and 4.

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

Notation 11. We denote the free monad F^* on F by T.

Property 10. *$F(X)$ is of the shape $\coprod_i R_i X \times S_i$, where*

- *S_i is an object of $\mathcal{C}$;*
- *R_i is a right adjoint functor*
- *its left adjoint L_i is such that $L_i K = \coprod_j K L'_{i,j}$ where $L'_{i,j}$ is an endofunctor on $\mathcal{D}$.*

Corollary 12. *$T(X)$ is isomorphic to $\coprod_i R_i T(X) \times S_i + X$.*

Proof. By Lambek's lemma [4], applied to the endofunctor $G - +X$. $\square$

4 Main result

Our main result is that a coequaliser diagram in Kl_T selecting objects in $\mathcal{D}^+$ either has no unifier, either has colimiting cocone.

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

Definition 13. Given a category E, let E^* be E extended freely with a terminal object.

Remark 14. Adding a terminal object results in adding a terminal cocone to all diagrams.

As a consequence, we have a following lemma.

Lemma 15. *Let J be a diagram in a category E. 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 E^*.*

Thus, we are going to work in Kl_T^* rather than Kl_T.

The following result is also useful.

Lemma 16. *Given a category E, the canonical functor $E \to E^*$ creates colimits.*

This has some 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 $\mathcal{C}$) are also coproducts in Kl^*_T.

Notation 17. We denote by $\top$ the terminal object and by ! any terminal morphism.

Here is our main result.

Theorem 18. *Let $Kl^*_{\mathcal{D}^+}$ be the full subcategory of Kl^*_T consisting of objects of $\mathcal{D}^+ \cup \{\top\}$. Then, $Kl^*_{\mathcal{D}^+}$ has coequalisers. Moreover, the inclusion $Kl^*_{\mathcal{D}^+} \to Kl^*_T$ 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 $\mathcal{C}$ or $\mathcal{D}^+$.*

In other words, any coequaliser diagram $A \rightrightarrows TB$ in Kl^*_T where A and B are in $\mathcal{D}^+$ has a colimit $B \to TC$ where $C \in \mathcal{D}^+ \cup \{\top\}$. Rephrasing it without the adjoined terminal object, this means that either such a coequaliser has a colimit which can be computed in $Kl^*_{\mathcal{D}^+}$, either there is no cocone at all.

5 Notations

We denote the identity morphism at an object x by 1_x.

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.

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 Kl_T and $\mathcal{C}$.

We denote the Kleisli composition of $f : A \to TB$ and $g : TB \to TC$ by $f[g] = g^* \circ f$.

6 Pruning phase

Here we want to compute a pushout diagram in Kl^*_T, where one branch is a coproduct of free morphisms.

$$\begin{tikzcd}
\coprod_i KA_i \arrow[r, "\coprod_i Kf_i"] \arrow[d, "{[g_i]}"'] & \coprod_i KB_i \arrow[r, "\eta"] & T\coprod_i KB_i \arrow[d, "{[u_i]}"] \\
X \arrow[rr, "\sigma"'] & & Z
\end{tikzcd}$$

where $g_i : KA_i \to X$ and $u_i \in \hom_{Kl^*_T}(KB_i, Z)$.

Question 19. *Can we prove in general that the pushout of a free morphism, if it exists, is a free morphism?*

We denote such a situation by

$$X \vdash f_1 := g_1, f_2 := g_2, \cdots \Rightarrow u_1, u_2, \ldots ; \sigma \dashv Z$$

abbreviated as

$$X \vdash \vec{f} := \vec{g} \Rightarrow \vec{u}; \sigma \dashv Z$$

or even

$$X \vdash (f_i)_i := g \Rightarrow u; \sigma \dashv Z$$

with $g = [g_i]$ and $u = [u_i]$.

Question 20. *Could we use Reddy's syntax, to differentiate the input/output? We need to know what his syntax is the internal language of.*

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

$$\frac{}{X \vdash \vec{()} := \vec{()} \Rightarrow \vec{()}; 1_X \dashv X}$$

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

$$\frac{}{\top \vdash \vec{f} := \vec{g} \Rightarrow \vec{!}; ! \dashv \top}$$

The pushout can be decomposed into smaller components.

$$\frac{X \vdash \vec{f} := \vec{g} \Rightarrow \vec{u}; \sigma \dashv Z \qquad Z \vdash \vec{f'} := \vec{g'}[\sigma] \Rightarrow \vec{u'}; \sigma' \dashv Z'}{X \vdash \vec{f}, \vec{f'} := \vec{g}, \vec{g'} \Rightarrow \vec{u}[\sigma'], \vec{u'}; \sigma' \circ \sigma \dashv Z'} \tag{1}$$

This follows from the stepwise construction of coequalisers (Lemma 41).

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 $\mathcal{D}$). Thus, we want to compute the pushout

$$\begin{tikzcd} KA \arrow[r, "Kf"] \arrow[d, "g"'] & KB \arrow[r, "\eta"] & TKB \\ TC & & \end{tikzcd}$$

Since $TC \simeq I + C + \coprod_i R_i TC \times S_i$ (Corollary 12) and KA is connected (Property 7), $KA \to TC$ factors through one of the following coproduct injections:

- $\eta : C \hookrightarrow TC$ (metavariable case)
- $in_i : R_i TC \times S_i \hookrightarrow TC$ (operation case)

In the next subsections, we discuss the different cases.

6.1 Case $KA \hookrightarrow C$

We want to compute the pushout of free morphisms

$$\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g"'] & KB \arrow[r, "\eta"] & TKB \\
C \arrow[d, "\eta"'] & & \\
TC & &
\end{tikzcd}$$

Since the functor $\mathcal{C} \to Kl_T$ is left adjoint, the pushout can be computed in $\mathcal{C}$. Now, C is in $\mathcal{D}^+$. Since KA is connected (Property 7), $KA \xrightarrow{g} C$ factors as $KA \xrightarrow{g'} KD \to KD + C' \cong C$. Therefore, we have to compute the pushout

$$\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d] & KB \\
KD \arrow[d] & \\
KD + C' &
\end{tikzcd}$$

which is the composition of pushouts

$$\begin{tikzcd}
KA \arrow[r] \arrow[d] & KB \arrow[d] \\
KD \arrow[r] \arrow[d] & KE \arrow[d] \\
KD + C' \arrow[r] & KE + C'
\end{tikzcd}$$

where E is given by Property 3

. Therefore, we have the rule

$$\frac{\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g'"'] & KB \arrow[d, "u"] \\
KD \arrow[r, "v"'] & KE
\end{tikzcd} \text{ is a pushout}}{KD + C' \vdash f := \eta \circ in_1 \circ g' \Rightarrow \eta \circ in_1 \circ u; T(v + C') \dashv KE + C'}$$

TODO: examples.

6.2 Case $KA \hookrightarrow R_iTC \times S_i$

We want to compute the pushout

$$\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g"'] & KB \arrow[r, "\eta"] & TKB \\
R_iTC \times S_i \arrow[d, "in_i"'] & & \\
TC & &
\end{tikzcd}$$

A cocone in Kl_T is given by an object Y with morphisms $KB \to TY \leftarrow C$ such that the following diagram commutes.

$$\begin{tikzcd}
KA \arrow[rr, "Kf"] \arrow[d, "g"'] & & KB \arrow[d] \\
R_iTC \times S_i \arrow[r] & R_iTY \times S_i \arrow[r] & TY
\end{tikzcd}$$

Exploiting the fact that both KA and KB are connected (Property 7) and the characterisation of TY as a coproduct $R_iTY \times S_i + \dots$ (Corollary 12), it follows that $KB \to TY$ factors through $R_iTY \times S_i \hookrightarrow TY$. Since $R_iTY \times S_i \to TY$ is monomorphic (as a coproduct injection, again by Property 6), a cocone in Kl_T is given by an object Y with morphisms $C \to TY$ and $KB \to R_iTY \times S_i$ such that the following diagram commutes.

$$\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g"'] & KB \arrow[d] \\
R_iTC \times S_i \arrow[r] & R_iTY \times S_i
\end{tikzcd}$$

which is equivalent to the commutation of the two following diagrams

$$\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g_2"'] & KB \arrow[d] \\
S_i \arrow[r, "="'] & S_i
\end{tikzcd}
\qquad
\begin{tikzcd}
KA \arrow[r, "Kf"] \arrow[d, "g_1"'] & KB \arrow[d] \\
R_iTC \arrow[r] & R_iTY
\end{tikzcd}$$

Uniqueness of $KB \to S_i$ making the left diagram commutes is ensured by the fact that Kf is epimorphic (Property 4). Thus, we already have the following rule in case no such morphism exists (this happens concretely for $M(x) := y$ when $y \neq x$).

$$\frac{g_2 \text{ does not factor through } Kf}{C \vdash f := in_i \circ (g_1, g_2) \Rightarrow !;! \dashv \top} \qquad (2)$$

In fact, using Property 5, the premise can be rephrased as an ineqality:

$$\frac{\mathrm{Ker}(Kf) \xrightarrow{p_1} KA \xrightarrow{g_2} S_i \neq \mathrm{Ker}(Kf) \xrightarrow{p_2} KA \xrightarrow{g_2} S_i}{C \vdash f := in_i \circ (g_1, g_2) \Rightarrow !; ! \dashv \top}$$

Remark 21. If S_i is orthogonal [1, Definition 1.32] to all morphisms (i.e., given any span $S_i \leftarrow A \rightarrow B$, there exists a unique $B \rightarrow S_i$ completing the triangle), as in the case where S_i is the output type "dirac", then this rule never applies.

Since R_i has a left adjoint L_i such that $L_i K = \coprod_j KL'_{i,j}$ (Property 10), the right diagram is equivalent to making the following diagram commute.

$$\begin{tikzcd}
\coprod_j KL'_{i,j}A \arrow[r, "\coprod_j KL'_{i,j}f"] \arrow[d, "g_1^*"'] & \coprod_j KL'_{i,j}B \arrow[d, "u"] \\
TC \arrow[r, "\sigma"'] & TY
\end{tikzcd}$$

Thus, this justifies the following rule

$$\frac{C \vdash (L'_{i,j}f)_j := g_1^* \Rightarrow u; \sigma \dashv Y \quad g_2 = h \circ Kf}{C \vdash f := in_i \circ (g_1, g_2) \Rightarrow in_i \circ u^*; \sigma \dashv Y} \tag{3}$$

7 Coequalising phase

Here we want to compute a coequalising diagram in Kl_T^*, where the domain is in $\mathcal{D}^+$.

$$\begin{tikzcd}
\coprod_i KA_i \arrow[r, shift left, "{[t_i]}"] \arrow[r, shift right, "{[u_i]}"'] & \Gamma \arrow[r, "\sigma"] & \Delta
\end{tikzcd}$$

where $t_i, u_i \in \hom_{Kl_T^*}(KB_i, \Gamma)$. We denote such a situation by

$$\Gamma \vdash t_1 =_{A_1} u_1, \dots t_p =_{A_p} u_p \Rightarrow \sigma \dashv \Delta$$

that we sometimes abbreviate as

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

The simplest case is when the coproduct is empty: then, the coequaliser 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

$$\frac{}{\top \vdash \vec{t} =_{\vec{A}} \vec{u} \Rightarrow ! \dashv \top}$$

Again, thanks to Lemma 41, such a coequaliser can be decomposed into smaller components.

$$\frac{\Gamma \vdash t_0 =_{n_0} u_0 \Rightarrow \sigma_0 \dashv \Delta_1 \qquad \Delta_1 \vdash \vec{t}[\sigma_0] =_{\vec{n}} \vec{u}[\sigma_0] \Rightarrow \sigma \dashv \Delta_2}{\Gamma \vdash t_0 =_{n_0} u_0, \vec{t} =_{\vec{n}} \vec{u} \Rightarrow \sigma_0[\sigma] \dashv \Delta_2} \tag{4}$$

Thanks to the previous rules, we can focus on the case where the coproduct is a singleton (since we focus on finite coproducts of elements of $\mathcal{D}$), and $\Gamma = TC$. Thus, we want to compute the coequaliser

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

Since $TC \simeq I + C + \coprod_i R_i TC$ (Corollary 12) and KA is connected (Property 7), $t, u : KA \to TC$ factor through one of the following coproduct injections:

- $\eta : C \hookrightarrow TC$ (metavariables)
- $in_i : R_i TC \times S_i \hookrightarrow TC$ (variables / operations)

In the next subsections, we discuss the different cases.

- $\eta = \ldots$, or $M(\vec{x}) = ...$, in case of a successful occur-check, in Section 7.1;
- $\eta = \eta$, or $M(\vec{x}) = M(\vec{y})$, in Section 7.2;
- $in_i = in_i$, or $o(\vec{t}) = o(\vec{u})$, in Section 7.3;

Let us mention schematically some other cases, which can never be unified in Kl_T, in most case by Property 7, and thus are solved using $\top$.

- $in_i = in_{i'}$ with $i \neq i'$ (in the examples, $o(\vec{t}) = o'(\vec{u})$ when $o \neq o'$)
- $\eta = \ldots$, or $M(\vec{x}) = u$, with failing occur-check (Section 8), i.e., when M appears deep in u.

7.1 Successful occur-check

The occur-check phase is described in more details in Section 8. Here, we assume that

1. $C \simeq KB + C'$ for some B, C';
2. $t : KA \to TC$ factors as $KA \xrightarrow{f} KB \xhookrightarrow{in_B} TC \simeq KB + \ldots$;
3. $u : KA \to TC$ factors as $KA \xrightarrow{g} TC' \xhookrightarrow{in_{TC'}} TC$.

In the examples, the second condition means that t is a metavariable, and the last condition means that this metavariable does not occur in u, i.e., the occur-check is successful.

Thus, the coequaliser

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

is a coequaliser (in Kl_T^*) of the shape

$$\begin{tikzcd} & Y \arrow[dr] & \\ X \arrow[ur] \arrow[dr] & & Y+Z \\ & Z \arrow[ur] & \end{tikzcd},$$

with $X = KA$, $Y = TKB$ and $Z = TC'$

Remark 22. There is a canonical isomorphism between the category of cocones over such a coequaliser diagram and the category of cocones over the pushout diagram $Y \leftarrow X \rightarrow Z$ exists.

Therefore, computing this coequaliser amounts to computing the pushout. We are thus in the situation of the pruning phase, and we can justify the rule

$$\frac{C' \vdash f := g \Rightarrow v; \sigma \dashv Z}{KB + C' \vdash in_B \circ f = in_{TC'} \circ g \Rightarrow [v, \sigma] \dashv Z}$$

7.2 Case $M(x_1, \ldots, x_m) =_q M(y_1, \ldots, y_m)$

Here we are in the situation where $t, u : KA \rightarrow TC$ factor as $t', u' : KA \rightarrow C$ through $\eta : C \rightarrow TC$. Note that since postcomposition with η is precisely the left adjoint (and thus cocontinuous) functor from $\mathcal{C}$ to Kl_T, it is enough to compute the coequaliser in $\mathcal{C}$ and then precompose it with η.

We assume the following

- $C \simeq KB + C'$ (in practice C is in $\mathcal{D}^+$),
- $t', u' : KA \rightarrow C$ both factor through $KB \hookrightarrow C$.

We want to compute the coequaliser of the free algebra morphisms.

$$\begin{tikzcd} & KB \arrow[dr] & \\ KA \arrow[ur, "Kf"] \arrow[dr, "Kg"'] & & KB+C' \\ & KB \arrow[ur] & \end{tikzcd}$$

Lemma 23. *The coequaliser of a diagram of this shape, in any category which has the involved colimits, is $X + C'$, where X is the coequaliser or Kf and Kg.*

Thanks to Property 3, we know that such a colimit exists in $\mathcal{D}^+$.

All the arguments in this section justify the following rule, whose premise can always be satisfied.

$$\dfrac{\begin{tikzcd} & KB \arrow[dr] & & \\ KA \arrow[ur, "Kf"] \arrow[dr, "Kg"'] & & KB+C' \arrow[r, "\sigma"] & X \text{ is a coequaliser} \\ & KB \arrow[ur] & & \end{tikzcd}}{KB+C' \vdash \eta \circ in_B \circ Kf = \eta \circ in_B \circ Kf \Rightarrow \eta \circ \sigma \dashv X}$$

7.3 Case $o(\vec{t}) = o(\vec{u})$

Here we assume that $t, u : KA \to TC$ factors as $(t', s_1), (u', s_2) : KA \to R_iTC \times S_i$ through $R_iTC \times S_i \hookrightarrow TC$. A cocone in Kl_T is given by an object Y with a morphism $C \xrightarrow{\sigma} TY$ such that the following diagram commutes.

$$\begin{tikzcd} KA \arrow[r, "{t',s_1}"] \arrow[d, "{u',s_2}"'] & R_iTC \times S_i \arrow[r, "R_i\sigma^* \times S_i"] & R_iTY \times S_i \arrow[d, "in_i"] \\ R_iTC \times S_i \arrow[r, "R_i\sigma^* \times S_i"'] & R_iTY \times S_i \arrow[r, "in_i"'] & TY \end{tikzcd}$$

Since $R_iTY \times S_i \to TY$ is monomorphic (as a coproduct injection, by Property 6), the above commutation is equivalent to commutation of the following diagram.

$$\begin{tikzcd} KA \arrow[r, "{t',s_1}"] \arrow[d, "{u',s_2}"'] & R_iTC \times S_i \arrow[d, "R_i\sigma^* \times S_i"] \\ R_iTC \times S_i \arrow[r, "R_i\sigma^* \times S_i"'] & R_iTY \times S_i \end{tikzcd}$$

which is equivalent to the commutation of the two following diagrams

$$\begin{tikzcd} KA \arrow[r, "s_1"] \arrow[d, "s_2"'] & S_i \arrow[d, "="] \\ S_i \arrow[r, "="'] & S_i \end{tikzcd} \qquad \begin{tikzcd} KA \arrow[r, "t'"] \arrow[d, "u'"'] & R_iTC \arrow[d, "R_i\sigma^*"] \\ R_iTC \arrow[r, "R_i\sigma^*"'] & R_iTY \end{tikzcd}$$

Clearly, if $s_1 \neq s_2$ there is no unifier so that we have the rule

$$\frac{s_1 \neq s_2}{C \vdash in_i \circ (t', s_1) = in_i \circ (u', s_2) \Rightarrow ! \dashv \top}$$

. We assume the contrary: $s_1 = s_2 (= s)$. Since R_i has a left adjoint L_i such that $L_i K = \coprod_j KL'_{i,j}$ (Property 10), the right one is equivalent to making the following diagram commute.

$$\begin{tikzcd}
\coprod_j KL'_{i,j}A \arrow[r, "t'^*"] \arrow[d, "u'^*"'] & TC \arrow[d, "\sigma^*"] \\
TC \arrow[r, "\sigma^*"'] & TY
\end{tikzcd}$$

Thus, this justifies the following rule

$$\frac{C \vdash t'^* = u'^* \Rightarrow \sigma \dashv Y}{C \vdash in_i \circ (t', s) = in_i \circ (u', s) \Rightarrow \sigma \dashv Y} \tag{5}$$

8 Occur-check

The occur-check allows to jump from the coequalising phase (Section 7) to the pruning phase (Section 6), whenever the metavariable appearing at the toplevel of the l.h.s does not appear in the r.h.s. On the other hand, if it appears on the r.h.s and is not top-level, then there is no unifier: this section is devoted to a proof of this fact, see Corollary 30.

We prove hits by working in the category of presheaves on $\mathcal{D}$.

Let $J : \mathcal{C} \to \hat{\mathcal{D}}$ mapping c to the nerve functor $\mathcal{C}(K-, c)$. Moreover, let $G : \hat{\mathcal{D}} \to \hat{\mathcal{D}}$ mapping P to $\coprod_i JS_i \times \prod_j P_{L_{i,j}-}$. Then, GJ is isomorphic to JF, i.e., the following square commutes up to isomorphism.

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "F"] \arrow[d, "J"'] & \mathcal{C} \arrow[d, "J"] \\
\hat{\mathcal{D}} \arrow[r, "G"'] & \hat{\mathcal{D}}
\end{tikzcd}$$

Note that J has a left adjoint (since it is continuous), but more importantly, it preserves coproducts (Property 7) and filtered colimits (Property 8). As a consequence, we following square commutes up to isomorphism.

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "T"] \arrow[d, "J"'] & \mathcal{C} \arrow[d, "J"] \\
\hat{\mathcal{D}} \arrow[r, "G^*"'] & \hat{\mathcal{D}}
\end{tikzcd}$$

Then, we can define the size of a morphism as a universal G-algebra morphism to the constant presheaf $\mathbb{N}$, or define the occur-check by induction as a G-algebra morphism from $\hom_{\mathcal{C}}(K-, T(B + C))$ to $\hom_{\mathcal{C}}(K-, TC) + 1$.

8.1 Preliminary lemma

There is a M-algebra structure $M\mathbb{N} \to \mathbb{N}$ on the constant presheaf $\mathbb{N}$ that adds one to the maximum of the arguments. For any $o : X \to \mathbb{N}$, this induces an algebra morphism $MX \xrightarrow{|-|_o} \mathbb{N}$, which basically computes the maximal depth of an element of MX, with custom depth for leaves in X as specified by o. In practice we will have X as the initial presheaf (then flatness does not depend on o), or the terminal presheaf.

Definition 24. Let $o : X \to \mathbb{N}$. A natural transformation $n : A \to MX$ is said *o-flat* if $A \to MX \xrightarrow{|-|_o} \mathbb{N}$ is constant. It is said *flat* if there exists o that makes it o-flat.

Example 25. Any natural transformation out of a representable presheaf is 0-flat.

The crucial lemma is the following.

Proposition 26. *Let $A \xrightarrow{n} M\emptyset$ be a flat morphism. Then, the following diagram is a pullback*

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

Proof. See Section B. □

As an example, if $\mathcal{D}$ 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 $t = u$.

Question 27. *Can we prove this more categorically by simply assuming that A is tiny?*

Corollary 28. *Let $A \xrightarrow{n} M1$ be a flat morphism. Then, the following diagram is a pullback*

$$\begin{tikzcd}
A + A \times_{M1} A \arrow[r] \arrow[d] & M(A+1) \arrow[d, "n^*"] \\
A \arrow[r, "n"'] & M1
\end{tikzcd}$$

Proof. Consider the functor $G' = G + 1$. By universal property, it precisely generates the monad $M' = M(1 + -)$, so we apply Proposition 26 for M' (any M'-flat morphism becomes M-flat by assigning the unit weight to the leaf in 1). □

Corollary 29. *Let $A \xrightarrow{n} M1$ be a flat morphism and $\sigma : B \to M1$ be another one. Then, the following diagram is a pullback*

$$\begin{tikzcd}
A \times_{M1} MB + A \times_{M1} A \arrow[r] \arrow[d] & M(A+B) \arrow[d, "{[u,\sigma]^*}"] \\
A \arrow[r, "u"'] & M1
\end{tikzcd}$$

Proof. Note that the right morphism decomposes as

$$\begin{tikzcd}
 & M(A+B) \arrow[d] \\
 & MM(A+1) \arrow[d, "\mu"] \\
A + A \times_{M1} A \arrow[r] \arrow[d] & M(A+1) \arrow[d] \\
A \arrow[r, "u"'] & M1
\end{tikzcd}$$

where we inserted the known pullback at the bottom. Since coproducts are pullback stable, it is enough to show that $(A \times_{M1} A) \times_{M(A+1)} M(A+B) = A \times_{M1} A$ and $A \times_{M1} MB = A \times_{M(A+1)} M(A+B)$, which we do in the following diagrams (all squares are pullbacks).

$$\begin{tikzcd}
A \times_{M1} A \arrow[r] \arrow[dd, equal] & A \arrow[rr] \arrow[d, equal] & & M(A+B) \arrow[d] \\
 & A \arrow[rr] \arrow[d, equal] & & MM(A+1) \arrow[d, "\mu"] \\
A \times_{M1} A \arrow[r] \arrow[d] & A \arrow[r, "in_1"'] & A + 1 + GM(A+1) \arrow[r, "\cong"'] & M(A+1) \arrow[d] \\
A \arrow[rrr, "u"'] & & & M1
\end{tikzcd}$$

$$\begin{tikzcd}
A \times_{M1} MB \arrow[r] \arrow[dd] & MB \arrow[r] \arrow[d] \arrow[rr, dashed, bend left] & M(A+B) \arrow[d] & 0+B \arrow[r] \arrow[d] \arrow[dd, dashed, bend right] \arrow[rr, dashed, bend left] & (A+B) \arrow[d] & B \arrow[r] \arrow[d] & B \arrow[d] \\
 & MM1 \arrow[r, "MMin_2"] \arrow[d, "\mu"'] & MM(A+1) \arrow[d, "\mu"] & M1 \arrow[r, "Min_2"] & M(A+1) & M1 \arrow[r, equal] \arrow[d, equal] & M1 \arrow[d, "Min_2"] \\
A \arrow[r, "u"'] \arrow[d] & M1 \arrow[r, "Min_2"'] & M(A+1) \arrow[d] & 0 \arrow[r] \arrow[d] & A \arrow[d] & M1 \arrow[r, "Min_2"'] & M(A+1) \\
A \arrow[rr, "u"'] & & M1 & M1 \arrow[r, "Min_2"] & M(A+1) & &
\end{tikzcd}$$

□

Corollary 30. *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 $Kd \to TKd$*

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

Proof. Applying J to the above diagram, we get

$$\begin{tikzcd}
yc \arrow[r] \arrow[d, "f"'] & M(yd+J\Gamma) \arrow[d, "{[u,\sigma]^*}"] \\
yd \arrow[r, "u"'] & M1
\end{tikzcd}$$

Now, u is flat (Example 25), so we by the previous corollary, the top morphism factors through the pullback
$yd \times_{M1} M(J\Gamma) + yd \times_{M1} yd \to M(yd{+}\Gamma)$. The conclusion easily follows. □

Question 31. *Can we do the whole algorithm by translating it in the presheaf world?*

9 Termination

In this section, we adapt the usual termination argument. Indeed, because objects of $\mathcal{D}$ are connected (Property 7), we can define without ambiguity the *size* $|\Gamma|$ of a *context* Γ (i.e., an element of $\mathcal{D}^+$) as the length of the coproduct:

Lemma 32. *Let $\coprod_{i\in I} A_i$ and $\coprod_{j\in J} B_j$ be isomorphic finite coproducts of connected objects. Then, I and J are isomorphic sets.*

Proof. Assume an isomorphism $\coprod_{i\in I} A_i \cong \coprod_{j\in J} B_j$. By connectedness, each $A_k \to \coprod_{j\in J} B_j$ factors (uniquely) through a coproduct injection $B_{\alpha(k)} \to \coprod_{j\in J} B_j$. This induces a map $\alpha : I \to J$. Conversely, we get a map $\beta : J \to I$. Now, the composition $A_k \to \coprod_{j\in J} B_j \cong \coprod_{i\in I} A_i$ which equals $A_k \to B_{\alpha(k)} \to A_{\beta(\alpha(k))} \to \coprod_{i\in I} A_i$ must be the coproduct injection $A_k \to \coprod_{i\in I} A_i$. By connectedness, it follows that $\beta(\alpha(k)) = k$. Therefore, $\beta \circ \alpha = id$, and by a symmetrical argument, $\alpha \circ \beta = id$.

We also need to define the size $||t||$ of a *term* $t : Kd \to \Delta$ as $|t|_0$, as defined in Section 7.1. □

Let us recall the usual argument.

First, we show termination of the pruning phase 6, and then termination of the coequalising phase 7.

9.1 Pruning phase

There are two recursives rules: (1) and (3).

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

Question 33. *Can we define it truly recursively?*

9.2 Coequalising phase

There are two recursives rules: (4) and (5). Contrary to the pruning phase, the first one is not structurally recursive since a substitution is applied to the argument.

Lemma 34. *If there is a finite derivation tree of* $\Gamma \vdash \vec{t} = \vec{u} \Rightarrow \sigma \dashv \Delta$, *and* $\Delta \neq \bot$, *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 35. *If there is a finite derivation tree of*

$$\Gamma \vdash \vec{f} := \vec{g} \Rightarrow \vec{u}; \sigma \dashv \Delta$$

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 (4), is the following lemma.

Lemma 36. *If* σ *is a renaming, then* $||t[\sigma]|| = ||t||$.

From these

10 Applications

10.1 Nominal sets (untyped)

10.2 Nominal sets, simply-typed

Let T be the set of simple types. Here we consider the category of functors $(\mathrm{Set} \times T)_m \to \mathrm{Set}$, where Set is the category of finite families indexed by T (it is equivalent to the comma category J/T, where $J : \mathbb{F} \to \mathrm{Set}$ is the canonical inclusion), and the indice $-_m$ means that we restrict to monomorphisms.

Note that any morphism $\sigma : \Gamma \vdash \tau \to \Gamma + \Delta \vdash \tau$ is the equaliser of the two obvious morphisms $\Gamma + \Delta \vdash \tau \rightrightarrows \Gamma + \Delta + \Delta \vdash \tau$.

Example 37. Consider the functor for λ-calculus:

$$\begin{aligned} F(X)_{\Gamma\vdash\sigma} &= y_{\sigma\vdash\sigma}(\Gamma \vdash \sigma) && \text{((variables))} \\ &+ \coprod_{\tau',\sigma'} X(\Gamma, \tau' \vdash \sigma') \times y_{\vdash\tau'\Rightarrow\sigma'}(\Gamma \vdash \sigma) && \text{((abstraction))} \\ &+ \coprod_{\tau'} X(\Gamma \vdash \tau' \Rightarrow \sigma) \times X(\Gamma \vdash \tau') && \text{((application))} \end{aligned}$$

. In other words,

$$\begin{aligned} F(X) = &\coprod_{\sigma} y_{\sigma\vdash\sigma} \\ &+ \coprod_{\tau',\sigma'} X(-, \tau' \vdash \sigma') \times y_{\vdash\tau'\Rightarrow\sigma'} \\ &+ \coprod_{\tau',\sigma} X(- \vdash \tau' \Rightarrow \sigma) \times X(- \vdash \tau') \times y_{\vdash\sigma} \end{aligned}$$

10.3 Meta-arities as sets

Consider $\mathbb{S}$ the category where objects are natural numbers and a morphism $n \to p$ is a subset of $\{0, \dots, p-1\}$ of cardinal n. For instance, $\mathbb{S}$ can be taken as subcategory of $\mathbb{F}_m$ consisting of strictly increasing injections, or as the subcategory of the augmented simplex category consisting of injective functions. Again, we can define the endofunctor for λ-calculs as before. Then, a metavariable takes as argument a set of available variables, rather than a list of distinct variables.

10.4 Linear syntax

Should work with the new condition on S_i

Take $\mathcal{C} = \mathrm{Set}$ and $\mathcal{D}$ the full subcategory of representable presheaves. Intuitively, given $X \in \mathrm{Set}$, the set X_n is the set of expressions with exactly n (distinct) variables. Then, we can consider the linear lambda-calculus, as an endofunctor on Set mapping X to $F(X)$ where $F(X)_n = y1 + \coprod_{p+q=n} X_p \times$

$X_q + (n+1) \times X_{n+1}$. Note that we could also specify a non-linear binder by replacing $(n+1) \times X_{n+1}$ with $\coprod_{p>n} \begin{pmatrix} p \\ n \end{pmatrix} X_p$. We could also have a non linear application by replacing $\coprod_{p+q=n} X_p \times X_q$ with $X_n \times X_n$.

Then, $F^*(0)$ is the linear lambda-calculus. $F^*(yn)$ is the syntax of linear λ-calculus extended with one n-ary metavariable applied to n (distinct) variables.

Note that $F(X)$ is of the shape $I + \coprod_i X_{p_{i,1}} \times \cdots \times X_{p_{i,m_i}} \times yn_i$ and each $X \mapsto X_{p_{i,1}} \times \cdots \times X_{p_{i,m_i}} \times yn_i$ is left adjoint to $X \mapsto X_{n_i} \times (yp_{i,1} + \cdots + yp_{i,m_i})$. No! But almost, i.e., if there exists a morphism $A \to X_p \times y_n$, then in fact $A = A_n yn$ and there exists a morphism $A_n \times yp \to X$, but the converse is false.

Remark 38. We could have done the non-linear version in this setting as well, but the abstract syntax is more convoluted (see the binomial coefficient) and metavariables must still be linear.

Example 39. linear lambda calculus, quantum lambda calculus

10.5 Polymorphic syntax

WIP. Let $J^+ : \mathbb{F} \to \mathrm{Set}$ denotes the functor mapping n to the $n + 1^{th}$ cardinal $\{0, \ldots, n\}$.

Let S be a nominal set with Sn the set of types taking free type variables in X.

Now, contexts are of the shape $n; \sigma_1, \ldots, \sigma_p \vdash \tau$, where $\sigma_i, \tau \in Sn$. Categorically,

We first consider the category of functors $\int^{n \in \mathbb{F}_m} \mathrm{Set} \times Tn \to \mathrm{Set}$ (this is the oplax colimit), i.e., a morphism between $n, \Gamma \vdash \tau$ and $n', \Gamma' \vdash \tau'$ is a monomorphism $\sigma : n \to n'$ and renamings $\Gamma[\sigma] \to \Gamma'$ such that $\tau[\sigma] = \tau'$.

Note that a morphism $\sigma : n|\Gamma \vdash \tau \to n+k|\Gamma[\sigma]+\Delta \vdash \tau[\sigma]$ is the equaliser of the two obvious morphisms $n+k|\Gamma[\sigma]+\Delta \vdash \tau[\sigma] \rightrightarrows n+k+k|\Gamma[\sigma]+\Delta+\Delta \vdash \tau[\sigma]$.

Example 40. Following [3], we specify System F with the functor

$$\begin{aligned}
F(X)_{n;\Gamma\vdash\sigma} &= V_{n;\Gamma\vdash\sigma} && \text{(variables)}\\
&+ \coprod_{\tau} X(n+1; \Gamma \vdash \tau)\delta_{\sigma,\forall\tau} && \text{(type abstraction)}\\
&+ \coprod_{\tau'} X(n; \Gamma \vdash \forall\tau') \times (\sigma' : Sn)\delta_{\sigma,\tau'[\sigma']} && \text{(type application)}\\
&+ \coprod_{\tau} X(n; \Gamma, \tau \vdash \sigma')\delta_{\sigma,\tau\Rightarrow\sigma'} && \text{(abstraction)}\\
&+ \coprod_{\tau'} X(n; \Gamma \vdash \tau' \Rightarrow \sigma) \times X(n; \Gamma \vdash \tau') && \text{(application)}
\end{aligned}$$

Here, V is defined as $V(n; \Gamma \vdash \sigma) = \#\{\sigma \in \Gamma\}$.

11 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] Makoto Hamana. Polymorphic abstract syntax via grothendieck construction. 2011.

[4] Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

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

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

A Some lemmas

Lemma 41. *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 $A_2 \rightrightarrows B \to C$.*

Lemma 42. *Let M be an algebraically free monad on an extensive category. Then, for any Kleisli morphism $B \to M(B')$, for any object A, the following square is a pullback (TODO: refine the assumptions)*

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & M(A+B) \arrow[d] \\
A \arrow[r] & M(A+B')
\end{tikzcd}$$

Proof. By Lambek's lemma, $M(X) \cong X + GMX$ where $M = G^*$. Through this isomorphism, this square becomes

$$\begin{tikzcd}
A \arrow[r] \arrow[d] & A + B + GM(A+B) \arrow[d] \\
A \arrow[r] & A + B' + GM(A+B')
\end{tikzcd}$$

where horizontal morphisms are left coproduct injections. TODO: conclude by extensivity. ☐

Lemma 43. *Let M be an algebraically free cartesian monad on an extensive category. Then, for any Kleisli morphism $B \to MB'$, the following square is a pullback (TODO: refine the assumptions)*

$$\begin{tikzcd}
MB \arrow[r] \arrow[d] & M(A+B) \arrow[d] \\
MB' \arrow[r] & M(A+B')
\end{tikzcd}$$

Proof. This commuting square expands as

$$\begin{tikzcd}
MB \arrow[r, "Min_2"] \arrow[d] & M(A+B) \arrow[d] \\
MMB' \arrow[r] \arrow[d, "="'] & M(A + M(A+B')) \arrow[d] \\
MMB' \arrow[r, "MMin_2"] \arrow[d, "\mu"'] & MM(A+B') \arrow[d, "\mu"] \\
MB' \arrow[r, "Min_2"'] & M(A+B')
\end{tikzcd}$$

where each square is a pullback square. ☐

Lemma 44. *Grothendieck topoi are stable under products.*

Proof. Let $(A_i)_{i \in I}$ be a family of Grothendieck topoi. Then, there exists a family of small categories $(B_i)_{i \in I}$ such that each A_i is reflective subcategory of $\hat{B}_i$ and moreover, the reflector is left exact. Then, $\prod_i A_i \to \prod_i \hat{B}_i \cong \widehat{\coprod_i B_i}$ is also coreflective and the reflector is left exact, concluding the proof. ☐

B Proof of Proposition 26

The results of this section are formalised in agda (see occurcheck-ind.agda). The crucial point that keeps the agda formalisation simple is that we can work on $[\mathcal{D}_0, \mathrm{Set}]$ rather than in $[\mathcal{D}, \mathrm{Set}]$, where $\mathcal{D}_0$ is the discrete category consisting of objects of $\mathcal{D}$.

Now, the basic idea is that we can define by recursion a morphism $MX \to M0 + \mathbb{N}$ such that the following is a pullback

$$\begin{tikzcd}
M0 \arrow[r] \arrow[d] & MX \arrow[d] \\
M0 \arrow[r] & M0 + \mathbb{N}
\end{tikzcd}$$

This morphism either returns the closed term is the given input $t \in MX$ is a closed term, or it returns the depth of some leaf. Assuming it indeed returns n, then the height of $t[u]$ must be bigger than the height of u raised by n.

Now, the basic idea for the proof is that if $t[u] = u$ then it must be that $n = 0$, so t is a metavariable (otherwise, it is a closed term).