Generic pattern unification: a categorical approach

Ambroise Lafont[0000−0002−9299−641X] and Neel Krishnaswami[0000−0003−2838−5865]

University of Cambridge

Abstract We provide a generic categorical setting for Miller's pattern unification. The syntax with metavariables is generated by a free monad applied to finite coproducts of representable functors; the most general unifier is computed as a coequaliser in the Kleisli category restricted to such coproducts. Our setting handles simply-typed second-order syntax, linear syntax, or (intrinsic) polymorphic syntax such as system F.

Keywords: Unification · Category theory · Syntax.

1 Introduction

Unification consists in finding a *unifier* of two terms t, u, that is a (metavariable) substitution σ such that $t[\sigma] = u[\sigma]$. Unification algorithms try to compute a most general unifier σ, in the sense that given any other unifier δ, there exists a unique δ' such that $\delta = \sigma[\delta']$.

First-order unification [19] is used in ML-style type inference systems and logic programming languages such as Prolog. For more advanced type systems, where variable binding is crucially involved, one needs second-order unification [12], which is undecidable [9]. However, Miller [15] identified a decidable fragment: in so-called *pattern unification*, metavariables are allowed to take distinct variables as arguments. In this situation, we can write an algorithm that either fails in case there is no unifier, either computes the most general unifier.

First-order unification has been explained from a lattice-theoretic point of view by Plotkin [4], and later categorically analysed in [20,8]. However, there is little work on understanding pattern unification algebraically, with the notable exception of [21], working with normalised terms of simply-typed λ-calculus. The present paper can be thought of as a generalisation of their work.

As an introduction, we start by presenting pattern unification in the case of pure λ-calculus in Section §1.1. In Section §1.2, we then present the generic algorithm summarised in Figure 1, instantiated for a syntax specified by a *binding signature*. Finally, in Section §1.3, we motivate our general setting and provide categorical semantics of the algorithm, by revisiting pure λ-calculus.

1.1 An example: pure $\boldsymbol{\lambda}$-calculus.

Consider the syntax of pure λ-calculus extended with metavariables satisfying the pattern restriction, encoded with De Bruijn levels, rather than De Bruijn indices [5]. More formally, the syntax is inductively generated by the following inductive rules, where C is a variable context $(0 : \tau, 1 : \tau, \ldots, n : \tau)$, often abbreviated as n, and τ denotes the sort of terms, which we often omit, while Γ is a metavariable context $M_1 : n_1, \ldots, M_m : n_m$ specifying a metavariable symbol M_i together with its number of arguments n_i.

$$\frac{x \in C}{\Gamma; C \vdash x} \qquad \frac{\Gamma; C \vdash t \quad \Gamma; C \vdash u}{\Gamma; C \vdash t\ u} \qquad \frac{\Gamma; C, |C| : \tau \vdash t}{\Gamma; C \vdash \lambda t}$$

$$\frac{M : n \in \Gamma \quad x_1, \ldots, x_n \in C \quad x_1, \ldots x_n \text{ distinct}}{\Gamma; C \vdash M(x_1, \ldots, x_n)}$$

Note that the De Bruijn level convention means that the variable bound in $\Gamma; C \vdash \lambda t$ is $|C|$, the length of the variable context C.

A *metavariable substitution* $\sigma : \Gamma \to \Gamma'$ assigns to each declaration $M : n$ in Γ a term $\Gamma'; n \vdash \sigma_M$. This assignation extends (through a recursive definition) to any term $\Gamma; C \vdash t$, yielding a term $\Gamma'; C \vdash t[\sigma]$. The base case is $M(x_1, \ldots, x_n)[\sigma] = \sigma_M[i \mapsto x_{i+1}]$, where $-[i \mapsto x_{i+1}]$ is variable renaming. Composition of substitutions $\sigma : \Gamma_1 \to \Gamma_2$ and $\sigma' : \Gamma_2 \to \Gamma_3$ is then defined as $(\sigma[\sigma'])_M = \sigma_M[\sigma']$.

A *unifier* of two terms $\Gamma; C \vdash t, u$ is a substitution $\sigma : \Gamma \to \Gamma'$ such that $t[\sigma] = u[\sigma]$. A *most general unifier* of t and u is a unifier $\sigma : \Gamma \to \Gamma'$ that uniquely factors any other unifier $\delta : \Gamma \to \Delta$, in the sense that there exists a unique $\delta' : \Gamma' \to \Delta$ such that $\delta = \sigma[\delta']$. We denote this situation by $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Gamma'$, leaving the variable context C implicit. Intuitively, the symbol $\Rightarrow$ separates the input and the output of the unification algorithm, which either returns a most general unifier, either fails when there is no unifier at all (for example, when unifying $t_1\ t_2$ with λu). To handle the latter case, we add[1] a formal error metavariable context $\bot$ in which the only term (in any variable context) is a formal error term !, inducing a unique substitution $! : \Gamma \to \bot$, satisfying $t[!] = !$ for any term t. For example, we have $\Gamma \vdash t_1\ t_2 = \lambda u \Rightarrow\ ! \dashv \bot$.

We generalise the notation (and thus the input of the unification algorithm) to lists of terms $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{u} = (u_1, \ldots, u_n)$ such that $\Gamma; C_i \vdash t_i, u_i$. Then, $\Gamma \vdash \vec{t} = \vec{u} \Rightarrow \sigma \dashv \Gamma'$ means that σ unifies each pair (t_i, u_i) and is the most general one, in the sense that it uniquely factors any other substitution that unifies each pair (t_i, u_i). As a consequence, we get the following *congruence* rule for application.

$$\frac{\Gamma \vdash t_1, t_2 = u_1, u_2 \Rightarrow \sigma \dashv \Delta}{\Gamma \vdash t_1\ t_2 = u_1\ u_2 \Rightarrow \sigma \dashv \Delta}$$

Unifying a list of term pairs $t_1, \vec{t_2} = u_1, \vec{u_2}$ can be performed sequentially by first computing the most general unifier σ_1 of (t_1, u_1), then applying the substitution to $(\vec{t_2}, \vec{u_2})$, and finally computing the most general unifier of the resulting list of term pairs: this is precisely the rule U-Split in Figure 1.

Thanks to this rule, we can focus on unification of a single term pair. The idea here is to recursively inspect the structure of the given terms, until reaching

[1] This trick will be justified from a categorical point of view in Section §3.

a metavariable application $M(x_1,\ldots,x_n)$ at top level on either hand side of $\Gamma, M : n \vdash t, u$. Assume by symmetry $t = M(x_1,\ldots,x_n)$, then three mutually exclusive situations must be considered:

1. M appears deeply in u
2. M appears in u at top level, i.e., $u = M(y_1,\ldots,y_n)$;
3. M does not appear in u;

In the first case, there is no unifier because the size of both hand sides can never match after substitution. This justifies the rule

$$\frac{u \neq M(\ldots) \qquad u_{|\Gamma} \neq \underline{\ldots}}{\Gamma, M : n \vdash M(\vec{x}) = u \Rightarrow\ ! \dashv \bot}$$

where $u_{|\Gamma} \neq \underline{\ldots}$ means that u does not restrict to the smaller metavariable context Γ, and thus that M does appear in u.

In the second case, we are unifying $M(\vec{x})$ with $M(\vec{y})$. The most general unifier substitutes M with $M'(z_1,\ldots,z_p)$, where $z_1, \ldots, z_p$ is the family of common positions i such that $x_i = y_i$. We denote[2] such a situation by $n \vdash \vec{x} = \vec{y} \Rightarrow \vec{z} \dashv p$. We therefore get the rule

$$\frac{n \vdash \vec{x} = \vec{y} \Rightarrow \vec{z} \dashv p}{\Gamma, M : n \vdash M(\vec{x}) = M(\vec{y}) \Rightarrow M \mapsto M'(\vec{z}) \dashv \Gamma, M' : p} \tag{1}$$

The last case is unification of $M(\vec{x})$ with some u such that M does not appear in u, i.e., u restricts to the smaller metavariable context Γ. We denote such a situation by $u_{|\Gamma} = \underline{u'}$, where u' is essentially u but considered in the smaller metavariable context Γ. In this case, the algorithm enters a *pruning phase*. To give an example, when unifying an application $t\ u$ with a metavariable $M(x_1,\ldots,x_n)$ which does not occur in t,u, two fresh metavariables M_1 and M_2 are created. Then, t is unified with $M_1(x_1,\ldots,x_n)$, outputing a unifier σ_1, and $u[\sigma_1]$ is unified with $M_2(x_1,\ldots,x_n)$, outputing a unifier σ_2. Eventually, M is replaced with $(M_1(\vec{x})\ M_2(\vec{x}))[\sigma_2]$, where σ_2 is the output unifier. We call it *pruning* because unifying u' and $M(\vec{x})$ when M does not occur in u' consists in removing all *outbound* variables in u', i.e., those that are not among the arguments $x_1,\ldots,x_n$ of the metavariable, by producing a substitution that restricts the arities of the metavariables occurring in u'.

Let us introduce a specific notation for this phase: $\Gamma \vdash u' :> M(\vec{x}) \Rightarrow v; \sigma \dashv \Delta$ means that σ is the output pruning substitution, and v is essentially $u'[\sigma][x_{i+1} \mapsto i]$, the term that the metavariable M ought to be substituted with. Note that the metavariable symbol M is fresh in this notation: it appears neither in Γ nor u', and is not in the domain of σ.

The output σ, v defines a substitution $(\Gamma, M : n) \to \Delta$ which can be characterised as the most general unifier of u' and $M(\vec{x})$. We thus have the rule

$$\frac{u_{|\Gamma} = \underline{u'} \qquad \Gamma \vdash u' :> M(\vec{x}) \Rightarrow v; \sigma \dashv \Delta}{\Gamma, M : n \vdash M(\vec{x}) = u \Rightarrow \sigma, M \mapsto v \dashv \Delta} \tag{2}$$

[2] The similarity with the above introduced notation is no coincidence: as we will see, both are (co)equalisers.

Note that M is indeed substituted by v, as hinted above. As before, we generalise the pruning phase to handle lists $\vec{u} = (u_1, \dots, u_n)$ of terms such that $\Gamma; C_i \vdash u_i$, and lists of pruning patterns $(\vec{x}_1, \dots, \vec{x}_n)$ where each $\vec{x_i}$ is a choice of distinct variables in C_i. Then, $\Gamma \vdash \vec{u} :> M_1(\vec{x}_1), \dots, M_n(\vec{x}_n) \Rightarrow \vec{v}; \sigma \dashv \Delta$ means that σ is the common pruning substitution, and v_i is essentially $u_i[\sigma][x_{i,j+1} \mapsto j]$, which M_i ought to be substituted with. Again, $(\sigma, \vec{v})$ define a substitution from $\Gamma, M_1 : |\vec{x}_1|, \dots, M_n : |\vec{x}_n|$ to Δ which can be characterised as the most general unifier of $\vec{u}$ and $M_1(\vec{x}_1), \dots M_n(\vec{x}_n)$.

We can then handle application as follows.

$$\frac{\Gamma \vdash t, u :> M_1(\vec{x}), M_2(\vec{x}) \Rightarrow v_1, v_2; \sigma \dashv \Delta}{\Gamma \vdash t\ u :> M(\vec{x}) \Rightarrow v_1\ v_2; \sigma \dashv \Delta} \tag{3}$$

As for unification (rule U-Split), pruning can be done sequentially, as in the rule P-Split in Figure 1. The usage of $+$ as a separator will be formally justified later in Remark 2: it intuitively enforces that the metavariables used in $\vec{f_2}$ are distinct from the metavariable used in f_1. Thanks to this sequential rule, we can focus on pruning a single term. The variable case is straightforward.

$$\frac{y = x_{i+1}}{\Gamma \vdash y :> M(\vec{x}) \Rightarrow i; 1_\Gamma \dashv \Gamma} \qquad \frac{y \notin \vec{x}}{\Gamma \vdash y :> M(\vec{x}) \Rightarrow !; ! \dashv \bot} \tag{4}$$

In λ-abstraction, the bound variable $|C|$ need not be pruned: we extend the list of allowed variables accordingly.

$$\frac{\Gamma \vdash t :> M_1(\vec{x}, |C|) \Rightarrow v; \sigma \dashv \Delta}{\Gamma \vdash \lambda t :> M(\vec{x}) \Rightarrow \lambda v; \sigma \dashv \Delta} \tag{5}$$

The remaining case consists in unifying $N(\vec{x})$ and $M(\vec{y})$, or equivalently, pruning a metavariable $N(x_1, \dots, x_n)$, whose arity must then be restricted to those positions in $y_1, \dots, y_m$. To be more precise, consider the family $z_1, \dots, z_p$ of common values in $x_1, \dots, x_n$ and $y_1, \dots, y_m$, so that $z_i = x_{l_i} = y_{r_i}$ for some lists $(l_1, \dots, l_p)$ and $(r_1, \dots, r_p)$ of distinct elements of $\{0, \dots, n{-}1\}$ and $\{0, \dots, m{-}1\}$ respectively. We denote[3] such a situation by $n \vdash \vec{x} :> \vec{y} \Rightarrow \vec{l}; \vec{r} \dashv p$. Then, the metavariable N is substituted with $N'(\vec{r})$ for some new metavariable N' of arity p, while the metavariable M is replaced with $N'(\vec{l})$:

$$\frac{n \vdash \vec{x} :> \vec{y} \Rightarrow \vec{l}; \vec{r} \dashv p}{\Gamma, N : n \vdash N(\vec{x}) :> M(\vec{y}) \Rightarrow N'(\vec{l}); N \mapsto N'(\vec{r}) \dashv \Gamma, N' : p} \tag{6}$$

This ends our description of the unification algorithm, in the specific case of pure λ-calculus. The goal of this paper is to generalise it, by parameterising the algorithm by a signature specifying a syntax.

[3] Again, the similarity with the pruning notation is no coincidence: as we will see, both are pullbacks.

1.2 First generalisation: parameterisation by a binding signature

As a first step, let us parameterise the unification algorithm by a binding signature [17]. A syntax is then specified by a set of symbols O together with a list of natural numbers $\vec{\alpha}_o$ for each $o \in O$ specifying the number of arguments (the size of the list) and the number of bound variables in each argument. For example, pure λ-calculus is specified by $O = \{app, lam\}$ with $\vec{\alpha}_{app} = (0, 0)$ and $\vec{\alpha}_{lam} = (1)$. The unification algorithm described in the previous section straightforwardly generalises to any syntax specified by a binding signature. Figure 1 summarises the generic algorithm that we will later interpret in a more general setting, where metavariable arguments are morphisms in a category. Since nothing enforces them to be lists[4], the vector notation is dropped for these arguments in the figure, but we still use it in the following specialisation to syntax specified by a binding signature.

In the rule U-RigRig, the expression $o(\vec{t})$ can be an operation or a variable, in which case $\vec{t}$ is the empty list. If o is an operation, the exact nature of $\vec{t}$ depends on the arity $\alpha_o = (n_1, \dots, n_p)$ of o: then $\vec{t}$ is a list of terms of size p and $\Gamma; |C| + n_i \vdash t_i$ for each $i \in \{1, \dots, p\}$, where C is the variable context of $o(\vec{t})$. The rigid case in the pruning phase consists in two rules P-Rig and P-Fail. Both are concerned with pruning a non metavariable term $\Gamma; C \vdash o(\vec{t})$. In the variable case, these two rules instantiate to (4). More precisely, if o is a variable in C, the side condition $o = \vec{x} \cdot o'$ means that $o = x_{o'+1}$. On the other hand, if o is an operation, then $\vec{x} \cdot o$ is defined as o and thus the rule P-Rig always applies with $o' = o$. If $\alpha_o = (n_1, \dots, n_p)$, then the notation $\mathcal{L}^+\vec{x}^o$ essentially unfolds to a list of the same size as α_o and whose i^{th} element is $M_i(\vec{x}, |C|, \dots, |C| + n_i - 1)$. For example, for pure λ-calculus, $\mathcal{L}^+\vec{x}^o = M_1(\vec{x}), M_2(\vec{x})$ in the application case, and $\mathcal{L}^+\vec{x}^o = M_1(\vec{x}, |C|)$ in the abstraction case, thus recovering the rules (3) and (5).

Note that the premises of the rules U-FlexFlex and P-Flex are not explicitly defined in figure 1, although for a syntax specified by a binding signature, they have the same meaning as in the previous section. In fact, the generic algorithm works in a more general setting, as we are going to explain in the next section, so that they need to be customised for each specific situation.

1.3 Categorification

In this section, we define the syntax of pure λ-calculus from a categorical point of view in order to motivate our general categorical setting. We then explain the semantics of the generic unification algorithm summarised in Figure 1.

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 $\underline{n} = \{0, \dots, n-1\}$. The action on morphisms of $\mathbb{F}_m$ means that these expressions support injective renamings.

[4] See Section §8.2 for an example where arguments are sets.

Pure λ-calculus defines such a functor Λ by $\Lambda_n = \{t \mid \cdot; n \vdash t\}$. It satisfies 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' defined by $\Lambda'_n = \{t \mid M : p; n \vdash t\}$ now satisfies the recursive equation $\Lambda'_n = \underline{n} + \Lambda'_n \times \Lambda'_n + \Lambda'_{n+1} + \mathrm{Inj}(p, n)$, where $\mathrm{Inj}(p, n)$ is the set of injections between the cardinal sets p and n, corresponding to a choice of arguments for the metavariable. In fact, $\mathrm{Inj}(p, n)$ 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 F the endofunctor on $[\mathbb{F}_m, \mathrm{Set}]$ mapping X to $I + X \times X + X_{-+1}$, where I is the functor mapping n to $\underline{n}$, the functor Λ can be characterised as the initial algebra for F, thus satisfying the recursive equation $\Lambda \cong F(\Lambda)$. In other words, Λ is the free F-algebra on the initial functor 0. On the other hand, Λ' is characterised as the initial algebra for $F(-) + yp$, where yp is the (representable) functor $\mathbb{F}_m(p, -) : \mathbb{F}_m \to \mathrm{Set}$, thus satisfying the recursive equation $\Lambda' \cong F(\Lambda') + yp$. In other words, Λ' is the free F-algebra on yp. Denoting T the free F-algebra monad, Λ is $T(0)$ and Λ' is $T(yp)$. Similarly, the functor $T(yp + yq)$ corresponds to extending the syntax with another metavariable of arity q.

In the view to abstracting pattern unification, these observations motivate considering functor 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[5] F on it. Then, the abstract definition of a syntax extended with metavariables is the free F-algebra monad T applied to a finite coproduct of representable functors.

To understand how a unification problem is stated in this general setting[6], let us first provide the familiar metavariable context notation with a formal meaning.

Notation 1. *We denote a finite coproduct* $\coprod_{i \in \{M, N, \ldots\}} yn_i$ *of representable functors by a* (metavariable) context $M : n_M, N : n_N, \ldots$

If $\Gamma = (M_1 : m_1, \ldots, M_p : m_p)$ and Δ are metavariable contexts, a Kleisli morphism $\sigma : \Gamma \to T\Delta$ is equivalently given (by the Yoneda Lemma and the universal property of coproducts) by a λ-term $\Delta; m_i \vdash \sigma_i$ for each $i \in \{1, \ldots, p\}$: this is precisely the data for a metavariable substitution $\Delta \to \Gamma$. Thus, Kleisli morphisms are nothing but metavariable substitutions. Moreover, Kleisli composition corresponds to composition of substitutions.

---

[5] In Section §2.2, we make explicit assumptions about this endofunctor for the unification algorithm to properly generalise.

[6] What follows is a generalisation of the first-order case explained in [20, Chapter 8], in the sense that we consider a free monad on a presheaf category, rather than on sets.

A unification problem can be stated as a pair of parallel Kleisli morphisms $yp \underset{u}{\overset{t}{\rightrightarrows}} T\Gamma$ where Γ is a metavariable context, corresponding to selecting a pair of terms $\Gamma; p \vdash t, u$. A unifier is nothing but a Kleisli morphism coequalising this pair. The property required by the most general unifier means that it is the coequaliser, in the full subcategory spanned by coproducts of representable functors. 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, as stated in Section §3.

With this in mind, we can give categorical semantics to the unification notation in Figure 1 as follows.

Notation 2. *We denote a coequaliser* $A \underset{u}{\overset{t}{\rightrightarrows}} \Gamma \overset{\sigma}{\dashrightarrow} \Delta$ *in a category* $\mathscr{B}$ *by* $\Gamma \vdash t =_{\mathscr{B}} u \Rightarrow \sigma \dashv \Delta$*, sometimes even omitting* $\mathscr{B}$*.*

This notation is used in the unification phase, taking $\mathscr{B}$ to be the Kleisli category of T restricted to coproducts of representable functors, and extended with an error object $\perp$ (as formally justified in Section §3), with the exception of the premise of the rule U-FlexFlex, where $\mathscr{B} = \mathscr{D}$ is the opposite category of $\mathbb{F}_m$. The latter corresponds to the above rule (1), whose premise precisely means that $p \xrightarrow{\vec{z}} n \underset{\vec{y}}{\overset{\vec{x}}{\rightrightarrows}} C$ is indeed an equaliser in $\mathbb{F}_m$.

Remark 1. In Notation 2, when A is a coproduct $yn_1 + \cdots + yn_p$, then t and u can be thought of as lists of terms $\Gamma; n_i \vdash t_i, u_i$, hence the vector notation used in various rules (e.g., U-Split). Moreover, the usage of comma as a list separator in the conclusion is formally justified by the notation $a + c \xrightarrow{f,g} b$ given morphisms $a \xrightarrow{f} b \xleftarrow{g} c$.

Note that the rule U-Split is in fact valid in any category (see [20, Theorem 9]). To formally understand the rule U-NoCycle which we have already introduced in the case of pure λ-calculus, see (2), let us provide the pruning notation with categorical semantics.

Notation 3. *We denote a pushout diagram in a category* $\mathscr{B}$ *as below left by the notation as below right, sometimes even omitting* $\mathscr{B}$*.*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "t"'] & \Gamma' \arrow[d, dashed, "u"] \\ \Gamma \arrow[r, dashed, "\sigma"'] & \Delta \arrow[ul, phantom, "\ulcorner", very near start] \end{tikzcd} \quad \Leftrightarrow \quad \Gamma \vdash_{\mathscr{B}} t :> f \Rightarrow u; \sigma \dashv \Delta$$

Similarly to Notation 2, this is used in Figure 1, taking $\mathscr{B}$ to be the Kleisli category of T restricted to coproducts of representable functors, and extended with an error object $\perp$, with the exception of the premise of the rule P-Flex, where $\mathscr{B} = \mathscr{D}$ is the opposite category of $\mathbb{F}_m$. The latter corresponds to the above

Unification Phase (Section §4)

– Structural rules

$$\frac{}{\Gamma \vdash () = () \Rightarrow 1_\Gamma \dashv \Gamma} \qquad \frac{}{\bot \vdash \vec{t} = \vec{u} \Rightarrow\, ! \dashv \bot}$$

$$\frac{\Gamma \vdash t_1 = u_1 \Rightarrow \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash \vec{t_2}[\sigma_1] = \vec{u_2}[\sigma_1] \Rightarrow \sigma_2 \dashv \Delta_2}{\Gamma \vdash t_1, \vec{t_2} = u_1, \vec{u_2} \Rightarrow \sigma_1[\sigma_2] \dashv \Delta_2}\text{U-S{\scriptsize PLIT}}$$

– Rigid-rigid (Section §4.1)

$$\frac{\Gamma \vdash \vec{t} = \vec{u} \Rightarrow \sigma \dashv \Delta}{\Gamma \vdash o(\vec{t}) = o(\vec{u}) \Rightarrow \sigma \dashv \Delta}\text{U-R{\scriptsize IG}R{\scriptsize IG}} \qquad \frac{o \neq o'}{\Gamma \vdash o(\vec{t}) = o'(\vec{u}) \Rightarrow\, ! \dashv \bot}$$

– Flex-*, no cycle (Section §4.2)

$$\frac{u_{|\Gamma} = \underline{u'} \qquad \Gamma \vdash u' :> M(x) \Rightarrow v; \sigma \dashv \Delta}{\Gamma, M : b \vdash M(x) = u \Rightarrow \sigma, M \mapsto v \dashv \Delta}\text{U-N{\scriptsize O}C{\scriptsize YCLE}} \quad + \text{ symmetric rule}$$

– Flex-Flex, same (Section §4.3)

$$\frac{b \vdash x =_{\mathscr{D}} y \Rightarrow z \dashv c}{\Gamma, M : b \vdash M(x) = M(y) \Rightarrow M \mapsto M'(z) \dashv \Gamma, M' : c}\text{U-F{\scriptsize LEX}F{\scriptsize LEX}}$$

– Flex-Rigid, cyclic (Section §4.4)

$$\frac{u = o(\vec{t}) \qquad u_{|\Gamma} \neq \underline{\ldots}}{\Gamma, M : b \vdash M(x) = u \Rightarrow\, ! \dashv \bot}\text{U-C{\scriptsize YCLIC}} \quad + \text{ symmetric rule}$$

Pruning phase (Section §5)

– Structural rules

$$\frac{}{\Gamma \vdash () :> () \Rightarrow (); 1_\Gamma \dashv \Gamma} \qquad \frac{}{\bot \vdash \vec{t} :> \vec{f} \Rightarrow\, !; ! \dashv \bot}$$

$$\frac{\Gamma \vdash t_1 :> f_1 \Rightarrow u_1; \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash \vec{t_2}[\sigma_1] :> \vec{f_2} \Rightarrow \vec{u_2}; \sigma_2 \dashv \Delta_2}{\Gamma \vdash t_1, \vec{t_2} :> f_1 + \vec{f_2} \Rightarrow u_1[\sigma_2], \vec{u_2}; \sigma_1[\sigma_2] \dashv \Delta_2}\text{P-S{\scriptsize PLIT}}$$

– Rigid (Section §5.1)

$$\frac{\Gamma \vdash \vec{t} :> \mathcal{L}^+ x^o \Rightarrow \vec{u}; \sigma \dashv \Delta \qquad o = x \cdot o'}{\Gamma \vdash o(\vec{t}) :> N(x) \Rightarrow o'(\vec{u}); \sigma \dashv \Delta}\text{P-R{\scriptsize IG}} \qquad \frac{o \neq x \cdot \ldots}{\Gamma \vdash o(\vec{t}) :> N(x) \Rightarrow\, !; ! \dashv \bot}\text{P-F{\scriptsize AIL}}$$

– Flex (Section §5.2)

$$\frac{c \vdash_{\mathscr{D}} y :> x \Rightarrow y'; x' \dashv d}{\Gamma, M : c \vdash M(y) :> N(x) \Rightarrow M'(y'); M \mapsto M'(x') \dashv \Gamma, M' : d}\text{P-F{\scriptsize LEX}}$$

Figure 1. Summary of the rules

rule (6) whose premise precisely means that the following square is a pullback in $\mathbb{F}_m$.

$$\begin{tikzcd} p \arrow[r, "l"] \arrow[d, "r"'] & n \arrow[d, "x"] \\ m \arrow[r, "y"'] & C \end{tikzcd}$$

Remark 2. Let us add a few more comments about Notation 3. First, note that if A is a coproduct $yn_1 + \cdots + yn_p$, then t and u can be thought of as lists of terms $\Gamma; n_i \vdash t_i$ and $\Gamma'; n_i \vdash u_i$. In fact, in the situations we will consider, f will be of the shape $yn_1 + \cdots + yn_p \xrightarrow{f_1+\cdots+f_p} ym_1 + \cdots + ym_p$. This explains our usage of $+$ as a list separator in the rule P-Split.

Plan of the paper

In Section §2, we present our categorical setting. In Section §3, we state the existence of the most general unifier as a categorical property. Then we describe the construction of the most general unifier, as summarised in Figure 1, starting with the unification phase (Section §4), the pruning phase (Section §5), the occur-check (Section §6). We finally justify completeness in Section §7. Applications are presented in Section §8.

General notations

Given $n \in \mathbb{N}$, we denote the set $\{0, \ldots n-1\}$ by $\underline{n}$. $\mathscr{B}^{op}$ denotes the opposite category of $\mathscr{B}$. 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 $f : A \to B$ and $g : A' \to B$, we denote the induced morphism $A + A' \to B$ by f, g. Coproduct injections $A_i \to \coprod_{i\in I} A_i$ are typically denoted by in_i. Let T be a monad on a category $\mathscr{B}$. We denote its unit by η, and its Kleisli category by Kl_T: the objects are the same as those of $\mathscr{B}$, and a Kleisli morphism from A to B is a morphism $A \to TB$ in $\mathscr{B}$. We denote the Kleisli composition of $f : A \to TB$ and $g : B \to TC$ by $f[g] : A \to TC$.

2 General setting

In our setting, syntax is specified as an endofunctor F on a category $\mathscr{C}$. We introduce conditions for the latter in Section §2.1 and for the former in Section §2.2. Finally, in Section §2.3, we sketch some examples.

2.1 Base category

We work in a full subcategory $\mathscr{C}$ 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 which has finite connected limits.

Example 1. In Section §1.2, we considered $\mathcal{A} = \mathbb{F}_m$ the category of finite cardinals and injections. Note that $\mathscr{C}$ is equivalent to the category of nominal sets [7].

Remark 3. The main property that justifies unification of two metavariables as an equaliser or a pullback in $\mathcal{A}$ is that given any metavariable context Γ, the functor $T\Gamma : \mathcal{A} \to \mathrm{Set}$ preserves them, i.e., $T\Gamma \in \mathscr{C}$. In fact, the argument works not only in the category of metavariable contexts and substitutions, but also in the (larger) category of objects of $\mathscr{C}$ and Kleisli morphisms between them. However, counter-examples can be found in the total Kleisli category. Consider indeed the unification problem $M(x, y) = M(y, x)$, in the example of pure λ-calculus. We can define[7] a functor P that does not preserve finite connected colimits such that $T(P)$ is the syntax extended with a binary commutative metavariable $M'(-, -)$. Then, the most general unifier, computed in the total Kleisli category, replaces M with P. But in the Kleisli category restricted to coproducts of representable functors, or more generally, to objects of $\mathscr{C}$, the coequaliser replaces M with a constant metavariable, as expected.

Remark 4. The category $\mathcal{A}$ is intuitively the category of metavariable arities. A morphism in this category can be thought of as data to substitute a metavariable $M : a$ with another. For example, in the case of pure λ-calculus, replacing a metavariable $M : m$ with a metavariable $N : n$ amounts to a choice of distinct variables $x_1, \dots, x_n \in \{0, \dots, m-1\}$, i.e., a morphism $\hom_{\mathbb{F}_m}(n, m)$.

Lemma 1. *$\mathscr{C}$ is closed under limits, coproducts, and filtered colimits. Moreover, it is cocomplete.*

Proof. Cocompleteness follows from [1, Remark 1.56], since $\mathscr{C}$ is the category of models of a limit sketch, and is thus locally presentable, by [1, Proposition 1.51].

For the claimed closure property, 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 same argument applies for coproducts and filtered colimits: they commute with finite connected limits [2, Example 1.3.(vi)].

By right continuity of the homset bifunctor, any representable functor is in $\mathscr{C}$ and thus the embedding $\mathscr{C} \to [\mathcal{A}, \mathrm{Set}]$ factors the Yoneda embedding $\mathcal{A}^{op} \to [\mathcal{A}, \mathrm{Set}]$.

Notation 4. *We set $\mathscr{D} = \mathcal{A}^{op}$ and denote the fully faithful Yoneda embedding as $\mathscr{D} \xrightarrow{K} \mathscr{C}$. We denote by $\mathscr{D}^+ \xrightarrow{K^+} \mathscr{C}$ the full subcategory of $\mathscr{C}$ consisting of*

[7] Define P_n as the set of two-elements sets of $\{0, \dots, n-1\}$.

finite coproducts of objects of $\mathscr{D}$. Moreover, we adopt Notation 1 for objects of $\mathscr{D}^+$, that is, a coproduct $\coprod_{i\in\{M,N,\dots\}} Ka_i$ is denoted by a (metavariable) context $M : a_M, N : a_N, \dots$

Remark 5. $\mathscr{D}^+$ is equivalent to the category of finite families of objects of $\mathcal{A}$. Thinking of objects of $\mathcal{A}$ as metavariable arities (Remark 4), $\mathscr{D}^+$ can be thought of as the category of metavariable contexts.

We now abstract the situation by listing a number of properties that we will use to justify the unification algorithm.

Property 1. The following properties hold.

(i) $\mathscr{D}$ has finite connected colimits.
(ii) $K : \mathscr{D} \to \mathscr{C}$ preserves finite connected colimits.
(iii) Given any morphism $f : a \to b$ in $\mathscr{D}$, the morphism Kf is epimorphic.
(iv) Coproduct injections $A_i \to \coprod_j A_j$ in $\mathscr{C}$ are monomorphisms.
(v) 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 $A_j \to \coprod_i A_i$.

Proof. (i) We assume that $\mathcal{A}$ has finite connected limits. Hence, its opposite category $\mathscr{D} = \mathcal{A}^{op}$ has finite connected colimits.

(ii) Let $y : \mathcal{A}^{op} \to [\mathcal{A}, \mathrm{Set}]$ denote the Yoneda embedding and $J : \mathscr{C} \to [\mathcal{A}, \mathrm{Set}]$ denote the canonical embedding, so that

$$y = J \circ K. \tag{7}$$

Now consider a finite connected limit $\lim F$ in $\mathcal{A}$. Then,

$$\begin{aligned}
\mathscr{C}(K \lim F, X) &\cong [\mathcal{A}, \mathrm{Set}](JK \lim F, JX) && (J \text{ is fully faithful})\\
&\cong [\mathcal{A}, \mathrm{Set}](y \lim F, JX) && (\text{By } (7))\\
&\cong JX(\lim F) && (\text{By the Yoneda Lemma.})\\
&\cong \lim(JX \circ F) && (X \text{ preserves finite connected limits})\\
&\cong \lim([\mathcal{A}, \mathrm{Set}](yF-, JX)] && (\text{By the Yoneda Lemma})\\
&\cong \lim([\mathcal{A}, \mathrm{Set}](JKF-, JX)] && (\text{By } (7))\\
&\cong \lim \mathscr{C}(KF-, X) && (J \text{ is full and faithful})\\
&\cong \mathscr{C}(\mathrm{colim}\ KF, X) &&\\
& && (\text{By left continuity of the hom-set bifunctor})
\end{aligned}$$

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

(iii) A morphism $f : a \to b$ is epimorphic if and only if the following square is a pushout [14, Exercise III.4.4]

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

We conclude by *(ii)*, because all morphisms in $\mathscr{D} = \mathcal{A}^{op}$ are epimorphic by assumption.

(iv) This follows from Lemma 1, because a morphism $f : A \to B$ is monomorphic if and only if the following square is a pullback

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

(v) This follows from coproducts being computed pointwise (Lemma 1), and representable functors being connected, by the Yoneda Lemma.

Remark 6. Continuing Remark 3, unification of two metavariables as pullbacks or equalisers in $\mathcal{A}$ crucially relies on Property 1.(ii), which holds because we restrict to functors preserving finite connected limits.

2.2 The endofunctor for syntax

We assume given an endofunctor F on $[\mathcal{A}, \mathrm{Set}]$ defined by

$$F(X)_a = \coprod_{o \in O_a} \prod_{j \in J_{o,a}} X_{L_{o,j,a}},$$

for some functors $O : \mathcal{A} \to \mathrm{Set}$, $J : (\int O)^{op} \to \mathbb{F}$ and $L : (\int J)^{op} \to \mathcal{A}$, , where

- $\mathbb{F}$ is the category of finite cardinals and any morphisms between them;
- $\int O$ denotes the category of elements of O whose objects are pairs of an object a of $\mathcal{A}$ and an element o in O_a, and morphisms between (a, o) and (a', o') are morphisms $f : a \to a'$ such that $O_f(o) = o'$;
- $\int J$ denotes the category of elements of $\int O \xrightarrow{J} \mathbb{F} \hookrightarrow \mathrm{Set}$. Objects are triples (a, o, j), where a is an object of $\mathcal{A}$, $o \in O_a$, and $j \in \{0, \ldots, J_{a,o} - 1\}$, and a morphism in $\int J$ between (a, o, j) and (a', o', j') is a morphism $f : a \to a'$ such that $o = O_f(o')$ and $j' = J_f(j)$.

Example 2. For pure λ-calculus where $\mathcal{A} = \mathbb{F}_m$, we have $O_n = \{a, l\} + \{v_i | 0 \leq i < n\}$, and $J_{v_i} = 0$, $J_a = 2$, $J_l = 1$, and $L_{n,o,j} = n+1$ is if $o = l$, or n otherwise.

We moreover assume that F restricts as an endofunctor on $\mathscr{C}$, i..e, that it maps functors preserving finite connected limits to functors preserving finite connected limits. This has the following consequence.

Lemma 2. *O preserves finite connected limits.*

Proof. O is isomorphic to $F(1)$, where 1 is the constant functor mapping everything to the singleton set $\{0\}$. Since 1 trivially preserves limits, it is in $\mathscr{C}$ and thus $F(1) \cong O$ also is.

In Section §2.4, we provide sufficient and necessary conditions on J and L for F to restrict as an endofunctor on $\mathscr{C}$.

Lemma 3. *F is finitary and generates a free monad that restricts to a monad T on $\mathscr{C}$. Moreover, TX is the initial algebra of $Z \mapsto X + FZ$, as an endofunctor on $\mathscr{C}$.*

Proof. F is finitary because filtered colimits commute with finite limits [14, Theorem IX.2.1] and colimits. The free monad construction is due to [18].

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

Notation 5. *Let $Kl_{\mathscr{D}^+}$ denote the full subcategory of Kl_T consisting of objects in $\mathscr{D}^+$. Moreover, we denote by $\mathcal{L}^+ : \mathscr{D}^+ \to Kl_{\mathscr{D}^+}$ the functor which is the identity on objects and postcomposes any morphism $A \to B$ by $\eta_B : B \to TB$, and by $\mathcal{L}$ the functor $\mathscr{D} \hookrightarrow \mathscr{D}^+ \xrightarrow{\mathcal{L}^+} Kl_{\mathscr{D}^+}$.*

Property 2. The functor $\mathscr{D} \xrightarrow{\mathcal{L}} Kl_{\mathscr{D}^+}$ preserves finite connected colimits.

Proof. $K : \mathscr{D} \to \mathscr{C}$ preserves finite connected colimits by Property 1.(ii). Thus, postcomposition with the left adjoint $\mathscr{C} \to Kl_T$ yields a functor $\mathscr{D} \to Kl_T$ preserving those colimits. The result follows because this functor factors as $\mathscr{D} \xrightarrow{\mathcal{L}} Kl_{\mathscr{D}^+} \hookrightarrow Kl_T$, where the right functor is full and faithful.

Notation 6. *Given $f \in \hom_{\mathscr{D}}(a, b)$, $u : Kb \to X$, we denote $u \circ Kf$ by $f \cdot u$.*

Given $a \in \mathscr{D}$, $o : Ka \to O$, we denote $\coprod_{j \in J_{a,o}} KL_{a,o,j}$ by $\overline{o}$. Given $f \in \hom_{\mathscr{D}}(b, a)$, we denote the induced morphism $\overline{f \cdot o} \to \overline{o}$ by f^o.

Lemma 4. *For any $X \in \mathscr{C}$, a morphism $Ka \to FX$ is equivalently given by a morphism $o \in Ka \to O$, and a morphism $f : \overline{o} \to X$.*

Proof. This follows from Property 1.(v).

Notation 7. *Given $o : Ka \to O$ and $\vec{t} : \overline{o} \to TX$, we denote the induced morphism $Ka \to FTX \hookrightarrow TX$ by $o(\vec{t})$, where the first morphism $Ka \to FTX$ is induced by Lemma 4.*

Let $\Gamma = (M_1 : a_1, \ldots, M_p : a_p) \in \mathscr{D}^+$ and $x \in \hom_{\mathscr{D}}(a, a_i)$, we denote the Kleisli composition $Ka \xrightarrow{\mathcal{L}x} Ka_i \xrightarrow{in_i} \Gamma$ by $M_i(x) \in \hom_{Kl_T}(Ka, \Gamma) = \hom_{\mathscr{C}}(Ka, T\Gamma)$.

Property 3. 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:

- $M_i(x)$ for some unique i and $x : a \to a_i$,
- $o(\vec{t})$ for some unique $o : Ka \to O$ and $\vec{t} : \overline{o} \to T\Gamma$.

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

Property 4. Let $\Gamma = M_1 : a_1, \ldots, M_n : a_n \in \mathscr{D}^+$ and $\sigma : \Gamma \to T\Delta$. Then, for any $o : Ka \to O$, $\vec{t} : \overline{o} \to T\Gamma$, $u : b \to a$, $i \in \{1, \ldots, n\}$, $x : a \to a_i$,

$$o(\vec{t})[\sigma] = o(\vec{t}[\sigma]) \qquad M_i(x)[\sigma] = x \cdot \sigma_i$$

$$u \cdot (o(\vec{t})) = (u \cdot o)(\vec{t} \circ u^o) \qquad u \cdot M_i(x) = M(x \circ u)$$

2.3 Examples

The following table sketches some examples among those detailed in Section §8. The shape of metavariable arities determine the objects of $\mathcal{A}$, as hinted by Remark 4.

	Metavariable arity	Operations (example)
Pure λ-calculus	$n \in \mathbb{F}_m$	See the introduction.
Simply-typed λ-calculus	$\underbrace{\tau_1, \ldots, \tau_n \vdash \tau_o}_{\text{simple types}}$	$\dfrac{\Gamma \vdash t : \tau_1 \Rightarrow \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash t\ u : \tau_2}$
System F	$m; \underbrace{\tau_1, \ldots, \tau_n \vdash \tau_o}_{< m \text{ type variables}}$	$\dfrac{m+1 \vert wk(\Gamma) \vdash t : \tau}{m \vert \Gamma \vdash \Lambda t : \Lambda \tau}\textsc{Type-Abstr}$ $\dfrac{m \vert \Gamma \vdash t : \Lambda \tau \qquad m \vdash \sigma}{m \vert \Gamma \vdash t[\sigma] : \tau[* \mapsto \sigma]}\textsc{Type-App}$

2.4 Restricting *F* as an endofunctor on $\mathscr{C}$

In this section, we provide sufficient and necessary conditions for the functor F to restrict as an endofunctor on $\mathscr{C}$. Let us first introduce some notations.

Notation 8. *Given a functor $F : D \to \mathscr{B}$, we denote the limit (resp. colimit) of F by $\int_{d:D} F(d)$ or $\lim F$ (resp. $\int^{d:D} F(d)$ or $\operatorname{colim} F$), and the canonical projection $\lim F \to Fd$ by p_d.*

We denote the set of connected components of a category D by $||D||$.

Remark 7. In all our examples (see [13, Section 8]), O is a coproduct $\coprod_{\ell \in V} O_\ell$ for some $O_\ell : \mathcal{A} \to \mathrm{Set}$ and $J_{a,in_\ell(o)} = \gamma_\ell$ for some finite cardinal γ_ℓ. In this case, L is equivalently given by a family of functors $H_{\ell,j} : \int O_\ell \to \mathrm{Set}$ for each $j \in \gamma_\ell$ by $L_{a,in_\ell(o),j} = H_{\ell,j}(a,o)$. Then,

$$F(X)_a \cong \coprod_{\ell \in V} \coprod_{o \in O_\ell} \prod_{j \in \gamma_\ell} X_{H_{\ell,j}(a,o)}$$

In this case there is a simple characterisation.

Lemma 5. *If O and J are in Remark 7, then F restricts as an endofunctor on $\mathscr{C}$ if and only if each $O_l \in \mathscr{C}$ and each $H_{\ell,j}$ preserves finite limits, or equivalently, the canonical morphism*

$$H_{\ell,j}(\lim d, x) \to \int_i H_{\ell,j}(d_i, O_\ell(p_i)(x))$$

is an isomorphism.

As we will see, this is a consequence of the following general proposition.

Proposition 1. *The functor F restricts as an endofunctor on $\mathscr{C}$ if and only if $O \in \mathscr{C}$ and one of the following conditions hold:*

(i) *for each object b of $\mathcal{A}$, the functor $(a, o) \mapsto \prod_{j \in J_{a,o}} \mathcal{A}(b, L_{a,o,j})$ from $(\int O)^{op}$ to* Set *preserves finite connected limits;*

(ii) *for each object b of $\mathcal{A}$, any functor $d : I \to O$ with I finite and connected, and for any $x \in O_{\lim d}$, the canonical morphism*

$$\prod_{j \in J_{\lim d,x}} \mathcal{A}(b, L_{\lim d,x,j}) \to \int_{i:I} \prod_{j \in J_{d_i, O_{p_i}(x)}} \mathcal{A}(b, L_{d_i, O_{p_i}(x), j})$$

is an isomorphism.

(iii) *$J : (\int O)^{op} \to \mathbb{F}$ preserves finitely connected colimits (i.e., it maps finitely connected limits in $\int O$ to colimits in $\mathbb{F}$) and for any $d : I \to O$ with I finite and connected, any $x \in O_{\lim d}$ and $\beta \in J_{\lim d,x}$, the canonical morphism in $\mathcal{A}$*

$$L_{\lim d,x,\beta} \to \int_{(i,j):\overline{\beta}} L_{d_i, O_{p_i}(x), j}$$

is an isomorphism, where $\overline{\beta}$ is the full subcategory of $\int J_{d-, O_{p\_}(x)}$ consisting of pairs (i, j) such that $\beta = J_{p_i}(j)$.

Remark 8. The colimit of a functor $d : I \to \mathrm{Set}$ can be computed as the set $|| \int d||$ of connected components of $\int d$. In the last condition of the above proposition, $\overline{\beta}$ is equivalently the connected component of $\int J_{d-, O_{p-}(x)}$ corresponding to the element in the colimit $\int^i J_{d_i, O_{p_i}(x)}$ which is mapped to $\beta \in J_{\lim d,x}$ by the canonical morphism.

We have already seen in Lemma 2 that if F restricts on $\mathscr{C}$, then $O \in \mathscr{C}$. We need a few technical lemmas in order to address the other statements.

Lemma 6 (Limits commute with dependent pairs). *Given functors $K : D \to \mathrm{Set}$ and $G : \int K \to \mathrm{Set}$, the following canonical morphism is an isomorphism*

$$\int_{d:D} \coprod_{x \in K(d)} G(d, x) \to \coprod_{\alpha \in \lim K} \int_d G(d, \alpha_d)$$

Proof. It is straightforward to check that both sets share the same universal property.

Lemma 7. *Let $F : \mathscr{B} \to \mathrm{Set}$ be a functor. A functor $G : D \to \int F$ is equivalently given by a functor $H : D \to \mathscr{B}$ and an element $x \in \lim(F \circ H)$, retrieving G as $Gd = (Hd, x_d)$.*

Proof. $\int F$ is isomorphic to the opposite of the comma category y/F, where $y : \mathscr{B}^{op} \to [\mathscr{B}, \mathrm{Set}]$ is the Yoneda embedding. The statement follows from the universal property of a comma category.

Lemma 8. *Let $F : \mathscr{B} \to \mathrm{Set}$ preserving a limit $\lim G$. Let $x \in F \lim G$, thus inducing a functor $G_x : D \to \int F$ by the previous lemma. Then, the limit of G_x is $(\lim D, x)$.*

Proof. Let $\mathscr{C}$ denote the full subcategory of $[\mathscr{B}, \text{Set}]$ of functors preserving $\lim G$. Note that $\int F$ is isomorphic to the opposite of the comma category K/F, where $K : \mathscr{B}^{op} \to \mathscr{C}$ is the Yoneda embedding, which preserves $\operatorname{colim} G$, by an argument similar to the proof of Property 1.(ii). The forgetful functor from a comma category L/R to the product of the categories creates colimits that L preserve.

Corollary 1. *Given categories I and $\mathscr{B}$ such that any functor $d : I \to \mathscr{B}$ has a limit, if a functor $F : \mathscr{B} \to \text{Set}$ preserves each such limit, then the following properties are equivalent for a functor $G : \int F \to \mathscr{B}'$.*

(i) G preserves limits over any $d' : I \to \int F$.
(ii) For any $d : I \to \mathscr{B}$ and $x \in F \lim d$, the canonical morphism $G(\lim d, x) \to \int_{i:I} G(d_i, Fp_i(x))$ is an isomorphism, where $p_i : \lim d \to d_i$ is the canonical projection.

Proof. $(i) \Rightarrow (ii)$ Define $d' : I \to \int F$ as $d'_i = (d_i, Fp_i(x))$. The implication follows from $\lim d' = (\lim d, x)$, by Lemma 8.

$(ii) \Rightarrow (i)$ By Lemma 7 and Lemma 2, there exists $d : I \to \mathscr{B}$ and $x \in F \lim d$ such that $d'_i = (d_i, Fp_i(x))$. Again, the implication follows from $\lim d' = (\lim d, x)$, by Lemma 8.

It is straightforward to check that both objects share the same universal property.

Lemma 9. *Given functors $K : D \to \text{Set}$ and $G : \int K \to \mathscr{B}$, the following canonical morphism is an isomorphism*

$$\int^{d:D} \coprod_{x \in K(d)} G(d, x) \to \int^{(d,x):\int K} G(d, x)$$

Lemma 10. *Given a functor $K : D \to \mathscr{B}$, the following canonical morphism is an isomorphism*

$$\coprod_{\beta \in ||D||} \int^{d:\beta} Kd \to \int^{d:D} Kd$$

where $||D||$ is the set of connected components of D.

We now have enough tools to address the proof of 1. Let us introduce auxiliary lemmas.

Definition 1. *We define the functor $M : (\int O)^{op} \to \mathscr{C}$ by*

$$M(a, o) = \coprod_{j \in J_{a,o}} KL_{a,o,j}.$$

Remark 9. Given any $X \in \mathscr{C}$ and object a of $\mathcal{A}$,

$$F(X)_a \cong \coprod_{o \in O_a} \mathscr{C}(M(a, o), X)$$

Lemma 11. *If F restricts as an endofunctor on $\mathscr{C}$, then the functor M preserves finite connected colimits.*

Proof. By Corollary 1 and Lemma 2, it is enough to show that given a functor $d : I \to \mathcal{A}$, where I is a finite connected category, the canonical morphism $\alpha_x : \int^i M_{d_i, O_{p_i}(x)} \to M_{\lim d, x}$ is an isomorphism, for any $x \in O_{\lim d}$.

Now, if $X \in \mathscr{C}$, then $F(X)_a$ is isomorphic to $\coprod_{o \in O_a} \mathscr{C}(M_{a,o}, X)$. Since then $F(X) \in \mathscr{C}$, the canonical morphism $F(X)_{\lim d} \to \int_i F(X)_{d_i}$ is an isomorphism. The codomain unfolds as

$$\begin{aligned}
\int_i F(X)_{d_i} &\cong \int_{i:I} \coprod_{o \in O_{d_i}} \mathscr{C}(M_{d_i,o}, X) && \text{(By Remark 9)} \\
&\cong \coprod_{o \in \int_i O_{d_i}} \int_{i:I} \mathscr{C}(M_{d_i,o_i}, X) && \text{(By Lemma 6)} \\
&\cong \coprod_{o \in O_{\lim d}} \int_{i:I} \mathscr{C}(M_{d_i,O_{p_i}(o)}, X) && \text{(By Lemma 2)} \\
&\cong \coprod_{o \in O_{\lim d}} \mathscr{C}(\int^{i:I} M_{d_i,O_{p_i}(o)}, X) \\
& && \text{(By left cocontinuity of the homset bifunctor)}
\end{aligned}$$

Existence of the colimit in $\mathscr{C}$ is ensured by Lemma 1. Now, the domain unfolds as

$$F(X)_{\lim d} \cong \coprod_{o \in O_{\lim d}} \mathscr{C}(M_{\lim d, o}, X)$$

Through these isomorphisms, the canonical morphism $F(X)_{\lim d} \to \int_i F(X)_{d_i}$ is

$$\coprod_{o \in O_{\lim d}} \mathscr{C}(M_{\lim d,o}, X) \xrightarrow{\coprod_{o \in O_{\lim d}} \mathscr{C}(\alpha_o, X)} \coprod_{o \in O_{\lim d}} \mathscr{C}(\int^{i:I} M_{d_i,O_{p_i}(o)}, X) \tag{8}$$

Because it is a bijection, each $\mathscr{C}(\alpha_o, X)$ also is. By the Yoneda lemma, each α_o is itself an isomorphism.

It is easy to see from the proof that the converse implication also holds.

Lemma 12. *If $O \in \mathscr{C}$ and the functor M preserves finite connected colimits, then F restricts as an endofunctor on $\mathscr{C}$.*

Proof. It easily follows from the fact that (8) is an isomorphism if and only if each α_o is.

Lemma 13. *If F restricts as an endofunctor on $\mathscr{C}$, then $J : (\int O)^{op} \to \mathbb{F}$ preserves finite connected colimits.*

Proof. By Corollary 1 and Lemma 2, it is enough to show that given a functor $d : I \to \mathcal{A}$, where I is a finite connected category, the canonical morphism $\alpha_x : \int^i J_{d_i,O_{p_i}(x)} \to J_{\lim d,x}$ is an isomorphism, for any $x \in O_{\lim d}$.

Given a set X, we denote by $\underline{X} : \mathcal{A} \to \mathrm{Set}$ the constant functor mapping anything to X. Since $\underline{X}$ preserves finite connected limits, $F(\underline{X})$ also does. Thus, we know that the canonical morphism $F(\underline{X})_{\lim d} \to \int_{i:I} F(\underline{X})_{d_i}$ is an isomorphism. The codomain unfolds as

$$\begin{aligned}
\int_{i:I} F(\underline{X})_{d_i} &\cong \int_{i:I} \coprod_{o \in O_{d_i}} X^{J_{d_i,o}} \\
&\cong \coprod_{o \in \int_i O_{d_i}} \int_{i:I} X^{J_{d_i,o_i}} && \text{(By Lemma 9)} \\
&\cong \coprod_{o \in O_{\lim d}} \int_{i:I} X^{J_{d_i,O_{p_i}(o)}} && \text{(By Lemma 2)} \\
&\cong \coprod_{o \in O_{\lim d}} X^{\int^{i:I} J_{d_i,O_{p_i}(o)}} \\
& \qquad\qquad \text{(By left cocontinuity of the homset bifunctor)}
\end{aligned}$$

The domain unfolds as

$$F(\underline{X})_{\lim d} \cong \coprod_{o \in O_{\lim d}} X^{J_{\lim d,o}}$$

Through these isomorphisms, the canonical morphism $F(\underline{X})_{\lim d} \to \int_{i:I} F(\underline{X})_{d_i}$ becomes

$$\coprod_{o \in O_{\lim d}} X^{J_{\lim d,o}} \xrightarrow{\coprod_{o \in O_{\lim d}} X^{\alpha_o}} \coprod_{o \in O_{\lim d}} X^{\int^{i:I} J_{d_i,O_{p_i}(o)}}$$

Because it is a bijection, each X^{α_o} also is. Since this is true for any set X, by the Yoneda lemma, each α_o is itself an isomorphism.

Lemma 14. *If F restricts as an endofunctor on $\mathscr{C}$, then for any $d : I \to O$ with I finite and connected, any $x \in O_{\lim d}$ and $\beta \in J_{\lim d,x}$, the canonical morphism in $\mathcal{A}$*

$$\alpha_\beta : L_{\lim d,x,\beta} \to \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j} \tag{9}$$

is an isomorphism, where $\overline{\beta}$ is full subcategory of $\left(\int J_{d-,O_{p_-}(x)}\right)^{op}$ consisting of pairs (i,j) such that $\beta = J_{p_i}(j)$.

Proof. By Lemma 11 and Corollary 1, the canonical morphism $\int^i M_{d_i,O_{p_i}(x)} \to M_{\lim d,x}$ is an isomorphism. The domain unfolds as

$$\begin{aligned}
\int^i M_{d_i,O_{p_i}(x)} &\cong \int^i \coprod_{j\in J_{d_i,O_{p_i}(x)}} KL_{d_i,O_{p_i(x)},j} \\
&\cong \int_{(i,j):\int J_{d-,O_{p-}(x)}} KL_{d_i,O_{p_i(x)},j} && \text{(By Lemma 9)} \\
&\cong \coprod_{\beta\in\int^i J_{d_i,O_{p_i}(x)}} \int^{(i,j):\overline{\beta}} KL_{d_i,O_{p_i(x)},j} && \text{(By Lemma 10)}
\end{aligned}$$

where $\overline{\beta}$ is the connected component of $\int J_{d_-,\alpha_-}$ corresponding to β by Remark 8. Since the domain category of $i \mapsto J_{d_-,\alpha-}$ is assumed to be finite, and any $J_{d,\alpha}$ is finite, $\int J_{d_-,\alpha_-}$ is finite, and thus $\overline{\beta}$ also is. Since K preserves finite connected limits by Property 1.(ii), we get

$$\begin{aligned}
\int^i M_{d_i,O_{p_i}(x)} &\cong \coprod_{\beta\in\int^i J_{d_i,O_{p_i}(x)}} K\int^{(i,j):\overline{\beta}} L_{d_i,O_{p_i(x)},j} \\
&\cong \coprod_{\beta\in J_{\int_i(d_i,O_{p_i}(x))}} K\int^{(i,j):\overline{\beta}} L_{d_i,O_{p_i(x)},j} && \text{(By Lemma 13)} \\
&\cong \coprod_{\beta\in J_{\lim d,x}} K\int^{(i,j):\overline{\beta}} L_{d_i,O_{p_i(x)},j} && \text{(By Lemma 8)}
\end{aligned}$$

Through this isomorphism, the canonical morphism $\int^i M_{d_i,O_{p_i}(x)} \to M_{\lim d,x}$ becomes

$$\coprod_{\beta\in J_{\lim d,x}} K\int^{(i,j):\overline{\beta}} L_{d_i,O_{p_i(x)},j} \xrightarrow{\coprod_{\beta\in J_{\lim d,x}} K\alpha_\beta} \coprod_{j\in J_{\lim d,x}} KL_{\lim d,x,j} \tag{10}$$

where α_β is considered as a morphism in $\mathscr{D} = \mathcal{A}^{op}$. Now, because coproducts are computed pointwise in $\mathscr{C}$ by Lemma 1, this is an isomorphism if and only if each $K\alpha_\beta$ is. Since K is full and faithful, this means that each α_β is an isomorphism.

The converse statement holds

Lemma 15. *If $O \in \mathscr{C}$, $J : (\int O)^{op} \to \mathbb{F}_m$ preserves finite connected colimits, and each α_β as defined in (9) is an isomorphism then F restricts as an endofunctor on $\mathscr{C}$.*

Proof. This follows easily from Lemma 12 and the fact that (10) is an isomorphism if and only if each α_β is.

We finally prove Proposition 1.

Proof (Proposition 1).

Assume that F restricts as an endofunctor on $\mathscr{C}$. Then, by Lemma 2, $O \in \mathscr{C}$. Moreover, by Lemma 13, $J : (\int O)^{op} \to \mathbb{F}$ preserves finite connected colimits and Proposition 1.(iii) holds by Lemma 14. Conversely, if $O \in \mathscr{C}$, J preserves finite connected colimits and Proposition 1.(iii) holds, then F restricts as an endofunctor on $\mathscr{C}$, by Lemma 15.

Assuming $O \in \mathscr{C}$, it remains to show that F restricts as an endofunctor on $\mathscr{C}$ if and only if Proposition 1.(i) or Proposition 1.(ii). Note that both are equivalent by Corollary 1.

Let us show that Proposition 1.(ii) is equivalent to

$$\alpha_\beta : L_{\lim d,x,\beta} \to \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j}$$

being an isomorphism, under the further assumption that J preserves finite connected colimits, as in Proposition 1.(iii) thus concluding the argument. First, $\alpha_\beta : L_{\lim d,x,\beta} \to \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j}$ is an isomorphism if and only if the induced natural transformation $\mathcal{A}(-, L_{\lim d,x,\beta}) \to \mathcal{A}(-, \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j})$ is an isomorphism, by the Yoneda lemma, that is, if for each object b of $\mathcal{A}$, the canonical morphism $\mathcal{A}(b, L_{\lim d,x,\beta}) \to \mathcal{A}(b, \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j})$ is an isomorphism. Since this is true for any β, this implies that

$$\prod_{\beta \in J_{\lim d,x}} \mathcal{A}(b, L_{\lim d,x,\beta}) \xrightarrow{\prod_{\beta \in J_{\lim d,x}} \mathcal{A}(b,\alpha_\beta)} \prod_{\beta \in J_{\lim d,x}} \mathcal{A}(b, \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j}) \tag{11}$$

is an isomorphism. In fact, the converse implication holds as well. Now, the codomain is

$$\prod_{\beta\in J_{\lim d,x}} \mathcal{A}(b, \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(x),j}) \cong \prod_{\beta\in J_{\lim d,x}} \int_{(i,j):\overline{\beta}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(By right continuity of the homset bifunctor)

$$\cong \prod_{\beta\in J_{\int_i (d_i,O_{p_i}(x))}} \int_{(i,j):\overline{\beta}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(By Lemma 8)

$$\cong \prod_{\beta\in \int^i J_{d_i,O_{p_i}(x)}} \int_{(i,j):\overline{\beta}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(J preserves finite connected colimits)

$$\cong \prod_{\beta\in ||J_{d_i,O_{p_i}(x)}||} \int_{(i,j):\overline{\beta}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(By Remark 8)

$$\cong \int_{(i,j):\int J_{d-,O_{p-}(x)}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(By Lemma 10)

$$\cong \int_{i:I} \prod_{j\in J_{d_i,O_{p_i}(x)}} \mathcal{A}(b, L_{d_i,O_{p_i}(x),j})$$

(By Lemma 9)

Therefore, (11) is an isomorphism if and only if the canonical morphism

$$\prod_{j\in J_{\lim d,x}} \mathcal{A}(b, L_{\lim d,x,j}) \to \int_{i:I} \prod_{j\in J_{d_i,O_{p_i}(x)}} \mathcal{A}(b, \int_{i:I} L_{d_i,O_{p_i}(x),j})$$

is an isomorphism, which is precisely what Proposition 1.(ii) states.

We can now address the simple situation of Remark 7.

Proof (Lemma 5). First, let us show that $O = \coprod_{\ell\in V} O_\ell$ preserves finite connected limits if and only if each O_ℓ does. By [2, Theorem 2.4 and Example 2.3.(iii)], since $\mathcal{A}$ has finite connected limits, a functor $G : \mathcal{A} \to \mathrm{Set}$ preserves those limits if and only if $\int G$ is a coproduct of filtered categories. The equivalence follows from $\int O = \coprod_{\ell\in V} \int O_\ell$.

Let us assume that O preserves finite connected limits: let us show that the other condition of Lemma 5 is equivalent to Lemma 14. First, $\int J$ is isomorphic to $\coprod_{\ell\in V} \gamma_\ell \times \int O_\ell$,, J preserves finite connected limits, by the same argument as above ($\int O$ has finite connected limits by Lemma 7 and Lemma 8).

Now, consider the canonical morphism $L_{\lim d,in_\ell(x),\beta} \to \int_{(i,j):\overline{\beta}} L_{d_i,O_{p_i}(in_\ell(x)),j}$, where $\overline{\beta}$ is the full subcategory of $\int J_{d-,O_{p\_}(x)}$ consisting of pairs (i,j) such that

$\beta = J_{p_i}(j)$. Because of the specific shape of J, this means $\beta = j$. Therefore, $\overline{\beta}$ is isomorphic to I, and this canonical morphism rewrites as

$$H_{\ell,\beta}(\lim d, x) \to \int_{i:I} H_{\ell,\beta}(d_i, O_{\ell,p_i}(x))$$

which is the same morphism that is considered in Lemma 5. It is an isomorphism if and only if $H_{\ell,\beta}$ preserves finite connected limits, by It Corollary 1.

3 Main result

The main point of pattern unification is that a pair of parallel morphisms in $Kl_{\mathscr{D}+}$ either has no unifier, or has a coequaliser. Working with this logical disjunction is slightly inconvenient; we rephrase it in terms of a true coequaliser by freely adding a terminal object.

Definition 2. *Given a category $\mathscr{B}$, let $\mathscr{B}^*$ be $\mathscr{B}$ extended freely with a terminal object.*

Notation 9. *We denote by $\perp$ the freely added terminal object in $\mathscr{B}^*$. Recall that ! denotes any terminal morphism.*

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

Lemma 16. *Let J be a diagram in a category $\mathscr{B}$. The following are equivalent:*

1. *J has a colimit as long as there exists a cocone;*
2. *J has a colimit in $\mathscr{B}^*$.*

Proof. Straightforward, because a colimit is defined as an initial cocone.

The following result is also useful.

Lemma 17. *Given a category $\mathscr{B}$, the canonical embedding functor $\mathscr{B} \to \mathscr{B}^*$ creates colimits.*

As a consequence,

1. whenever the colimit in $Kl^*_{\mathscr{D}+}$ is not $\perp$, it is also a colimit in $Kl_{\mathscr{D}+}$;
2. existing colimits in $Kl_{\mathscr{D}+}$ are also colimits in $Kl^*_{\mathscr{D}+}$;
3. in particular, coproducts in $Kl_{\mathscr{D}+}$ (which are computed in $\mathscr{C}$) are also coproducts in $Kl^*_{\mathscr{D}+}$.

The main point of pattern unification is the following result.

Theorem 1. *$Kl^*_{\mathscr{D}+}$ has coequalisers.*

In the next sections, we show how the generic unification algorithm summarised in Figure 1 provides a construction of such coequalisers.

4 Unification phase

In this section, we describe the main unification phase, which computes a coequaliser in $Kl^*_{\mathscr{D}^+}$. We denote a coequaliser $\coprod_i Ka_i \overset{\vec{t}}{\underset{\vec{u}}{\rightrightarrows}} \Gamma \xrightarrow{\sigma} \Delta$ in $Kl^*_{\mathscr{D}^+}$ by $\Gamma \vdash \vec{t} = \vec{u} \Rightarrow \sigma \dashv \Delta$, following Notation 2.

Let us start with the structural rules. When $\Gamma = \bot$, the coequaliser is the terminal cocone, i.e., $\bot \vdash \vec{t} = \vec{u} \Rightarrow\, ! \dashv \bot$ holds. When the coproduct is empty, the coequaliser is just Γ, i.e., $\Gamma \vdash () = () \Rightarrow 1_\Gamma \dashv \Gamma$ holds.

Let us explicitly state the categorical involved lemma for the rule U-Split.

Lemma 18 (Theorem 9, [20]). *In any category, denoting morphism composition $f \circ g$ by $g[f]$, the following rule applies.*

$$\frac{\Gamma \vdash t_1 = u_1 \Rightarrow \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash t_2[\sigma_1] = u_2[\sigma_1] \Rightarrow \sigma_2 \dashv \Delta_2}{\Gamma \vdash t_1, t_2 = u_1, u_2 \Rightarrow \sigma_1[\sigma_2] \dashv \Delta_2}\,\textsc{U-Split}$$

In other words, if the first two diagrams below are coequalisers, then the last one as well

$$\begin{tikzcd} A_1 \arrow[r, bend left, "t_1"] \arrow[r, bend right, "u_1"'] & \Gamma \arrow[r, dashed, "\sigma_1"] & \Delta_1 \end{tikzcd} \qquad \begin{tikzcd} & \Gamma \arrow[dr, "\sigma_1"] & & \\ A_2 \arrow[ur, "t_2"] \arrow[dr, "u_2"'] & & \Delta_1 \arrow[r, dashed, "\sigma_2"] & \Delta_2 \\ & \Gamma \arrow[ur, "\sigma_1"'] & & \end{tikzcd}$$

$$A_1 + A_2 \overset{t_1, t_2}{\underset{u_1, u_2}{\rightrightarrows}} \Gamma \overset{\sigma_2 \circ \sigma_1}{\dashrightarrow} \Delta_2$$

Thanks to this rule which constructs coequalisers sequentially, we can now focus on the rules dealing with singleton lists, that is, with coequaliser diagrams $Ka \overset{t}{\underset{u}{\rightrightarrows}} T\Gamma$. By Property 3, $t, u : Ka \to T\Gamma$ are either rigid or flexible. In the next subsections, we discuss all the different mutually exclusive situations (up to symmetry):

- both t or u are rigid (Section §4.1),
- $t = M(\dots)$ and M does not occur in u (Section §4.2),
- t and u are $M(\dots)$ (Section §4.3),
- $t = M(\dots)$ and M occurs deeply in u (Section §4.4).

4.1 Rigid-rigid

Here we detail unification of $o(\vec{t})$ and $o'(\vec{u})$ for some $o, o' : Ka \to O$, morphisms $\vec{t} : \overline{o} \to T\Gamma$, $\vec{u} : \overline{o'} \to T\Gamma$.

Assume given a unifier $\sigma : \Gamma \to \Delta$. By Property 4, $o(\vec{t}[\sigma]) = o'(\vec{u}[\sigma])$. By Property 3, this implies that $o = o'$, $\vec{t}[\sigma] = \vec{u}[\sigma]$. Therefore, we get the following failing rule

$$\frac{o \neq o'}{\Gamma \vdash o(\vec{t}) = o'(\vec{u}) \Rightarrow\ ! \dashv \bot}$$

We now assume $o = o'$. Then, $\sigma : \Gamma \to \Delta$ is a unifier if and only if it unifies $\vec{t}$ and $\vec{u}$. This induces an isomorphism between the category of unifiers for $o(\vec{t})$ and $o(\vec{u})$ and the category of unifiers for $\vec{t}$ and $\vec{u}$, justifying the rule U-RigRig.

4.2 Flex-*, no cycle

Here we detail unification of $M(x)$, which is nothing but $\mathcal{L}x[in_M]$, and $u : Ka \to T(\Gamma, M : b)$, such that M does not occur in u, in the sense that $u = u'[in_\Gamma]$ for some $u' : Ka \to T\Gamma$. We exploit the following general lemma, recalling Notation 3.

Lemma 19 ([3], Exercise 2.17.1). *In any category, denoting morphism composition $g \circ f$ by $f[g]$, the following rule applies:*

$$\frac{\Gamma \vdash t :> t' \Rightarrow v; \sigma \dashv \Delta}{\Gamma + B \vdash t[in_1] = t'[in_2] \Rightarrow \sigma, v \dashv \Delta}$$

In other words, if the below left diagram is a pushout, then the below right diagram is a coequaliser.

$$\begin{tikzcd} A \arrow[r, "t'"] \arrow[d, "t"'] & B \arrow[d, dashed, "v"] \\ \Gamma \arrow[r, dashed, "\sigma"'] & \Delta \end{tikzcd} \qquad \begin{tikzcd} & \Gamma \arrow[rd, bend left, "in_1"] & & \\ A \arrow[ru, bend left, "t'"] \arrow[rd, bend right, "t"'] & & \Gamma + B \arrow[r, dashed, "\sigma,v"] & \Delta \\ & B \arrow[ru, bend right, "in_2"'] & & \end{tikzcd}$$

Taking $t = M(x) = \mathcal{L}x : Ka \to (M : b)$ and $t' = u'$, we thus have the rule

$$\frac{\Gamma \vdash u' :> M(x) \Rightarrow v; \sigma \dashv \Delta \qquad u = u'[in_\Gamma]}{\Gamma, M : b \vdash M(x) = u \Rightarrow \sigma, M \mapsto v \dashv \Delta} \tag{12}$$

Let us make the factorisation assumption about u more effective. We can define by recursion a partial morphism from $T(\Gamma, M : b)$ to $T\Gamma$ that tries to compute u' from an input data u.

Lemma 20. *There exists $m_{\Gamma;b} : T(\Gamma, M : b) \to T\Gamma + 1$ such that a morphism $u : Ka \to T(\Gamma, M : b)$ factors as $Ka \xrightarrow{u'} T\Gamma \hookrightarrow T(\Gamma, M : b)$ if and only if $m_{\Gamma;b} \circ u = in_1 \circ u'$.*

Proof. We construct m by *recursion*, by equipping $T\Gamma + 1$ with an adequate F-algebra. Considering the embedding $(\Gamma, M : b) \xrightarrow{\eta + !} T\Gamma + 1$, we then get the desired morphism by universal property of $T(\Gamma, M : b)$ as a free F-algebra. The claimed property is proven by induction (see the induction lemma 24 introduced later).

Therefore, we can rephrase (12) as the rule U-NoCycle in Figure 1, using the following notations (in the second one, we take Γ as the empty context).

Notation 10. *Given $u : Ka \to T(\Gamma, M : b)$, we denote $m_{\Gamma;b} \circ u$ by $u_{|\Gamma}$. Moreover, for any $u' : Ka \to T\Gamma$, we denote $in_1 \circ u' : Ka \to T\Gamma + 1$ by $\underline{u'}$.*

Notation 11. *Let Γ and Δ be metavariable contexts and $a \in \mathscr{D}$. Any $t : Ka \to T(\Gamma + \Delta)$ induces a Kleisli morphism $(\Gamma, M : a) \to T(\Gamma + \Delta)$ which we denote by $M \mapsto t$.*

4.3 Flex-Flex, same metavariable

Here we detail unification of $M(x) = \mathcal{L}x[in_M]$ and $M(y) = \mathcal{L}y[in_M]$, with $x, y \in \hom_{\mathscr{D}}(a, b)$. We exploit the following lemma with $u = \mathcal{L}x$ and $v = \mathcal{L}y$.

Lemma 21. *In any category, denoting morphism composition $g \circ f$ by $f[g]$, the following rule applies:*

$$\frac{B \vdash u = v \Rightarrow h \dashv C}{B + D \dashv u[in_B] = v[in_B] \Rightarrow h + 1_D \dashv C + D}$$

In other words, if the below left diagram is a coequaliser, then so is the below right diagram.

$$\begin{tikzcd} A \arrow[r, shift left, "u"] \arrow[r, shift right, "v"'] & B \arrow[r, dashed, "h"] & C \end{tikzcd} \qquad \begin{tikzcd} & B \arrow[dr, "in_B"] & & \\ A \arrow[ur, "u"] \arrow[dr, "v"'] & & B + D \arrow[r, dashed, "h+1_D"] & C + D \\ & B \arrow[ur, "in_B"'] & & \end{tikzcd}$$

It follows that it is enough to compute the coequaliser of $\mathcal{L}x$ and $\mathcal{L}y$. Furthermore, by Property 1.(i) and Property 2, it can be computed as the image of the coequaliser of x and y, thus justifying the rule U-FlexFlex, using Notation 11.

4.4 Flex-rigid, cyclic

Here we handle unification of $M(x)$ for some $x \in \hom_{\mathscr{D}}(a, b)$ and $u : Ka \to \Gamma, M : b$, such that u is rigid and M occurs in u, i.e., $\Gamma \to \Gamma, M : b$ does not factor u. In Section §6, we show that in this situation, there is no unifier. Then, Lemma 20 and Notation 10 justify the rule U-Cyclic.

5 Pruning phase

The pruning phase computes a pushout in $Kl^*_{\mathscr{D}+}$ of a span $\Gamma \xleftarrow{\vec{t}} \coprod_i Ka_i \xrightarrow{\coprod_i \mathcal{L}x_i} \coprod_i Kb_i$. We always implicitly assume (and enforce) that the right branch is a finite coproduct of free morphisms.

Remark 10. A pushout cocone for the above span consists in morphisms $\Gamma \xrightarrow{\sigma} T\Delta \xleftarrow{\vec{u}} \coprod_i Kb_i$ such that $\vec{t}[\sigma] = \vec{u} \circ \coprod_i Kx_i$, i.e., $t_i[\sigma] = \vec{x_i} \cdot u_i$ for each i.

When $\Gamma \xrightarrow{\sigma} \Delta \xleftarrow{\vec{u}} \coprod_i Kb_i$ is a pushout of the above span, we use Notation 3 and denote such a situation by $\Gamma \vdash \vec{t} :> \coprod_i \mathcal{L}x_i \Rightarrow \vec{u}; \sigma \dashv \Delta$.

Let us start with the structural rules. When $\Gamma = \bot$, the pushout is the terminal cocone, i.e., $\bot \vdash \vec{t} :> \vec{f} \Rightarrow !; ! \dashv \bot$ holds. When the coproduct is empty, the pushout is just Γ, i.e., $\Gamma \vdash () :> () \Rightarrow (); 1_\Gamma \dashv \Gamma$ holds. Finally, let us note that the sequential rule P-Split is in fact valid in any category.

Lemma 22. *In any category, denoting morphism composition $f \circ g$ by $g[f]$, the following rule applies.*

$$\frac{\Gamma \vdash t_1 :> f_1 \Rightarrow u_1; \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash t_2[\sigma_1] :> f_2 \Rightarrow u_2; \sigma_2 \dashv \Delta_2}{\Gamma \vdash t_1, t_2 :> f_1 + f_2 \Rightarrow u_1[\sigma_2], u_2; \sigma_1[\sigma_2] \dashv \Delta_2}\text{P-Split}$$

In other words, if the first two diagrams below are pushouts, then the last one as well.

$$\begin{tikzcd}
A \arrow[r, "f_1"] \arrow[d, "t_1"'] & A' \arrow[d, dashed, "u_1"] \\
\Gamma \arrow[r, dashed, "\sigma_1"'] & \Delta_1
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, "f_2"] \arrow[d, "t_2"'] & B' \arrow[dd, dashed, "u_2"] \\
\Gamma \arrow[d, "\sigma_1"'] & \\
\Delta_1 \arrow[r, dashed, "\sigma_2"'] & \Delta_2
\end{tikzcd}
\qquad
\begin{tikzcd}
A + B \arrow[r, "f_1 + f_2"] \arrow[d, "t_1, t_2"'] & A' + B' \arrow[d, dashed, "\sigma_2 \circ u_1, u_2"] \\
\Gamma \arrow[r, dashed, "\sigma_2 \circ \sigma_1"'] & \Delta_2
\end{tikzcd}$$

Thanks to the above rule, we can now focus on the case where $\vec{t}$ is a singleton list, thus dealing with a span $T\Gamma \xleftarrow{t} Ka \xrightarrow{N(x)} T(N : b)$. By Property 3, the left morphism $Ka \to T\Gamma$ is either flexible or rigid. Each case is handled separately in the following subsections.

5.1 Rigid

Here, we describe the construction of a pushout of $\Gamma \xleftarrow{o(\vec{t})} Ka \xrightarrow{N(x)} N : b$ where $o : Ka \to O$ and $\vec{t} : \overline{o} \to T\Gamma$. By Remark 10, a cocone is a cospan $T\Gamma \xrightarrow{\sigma} T\Delta \xleftarrow{t'} Kb$ such that $o(\vec{t})[\sigma] = x \cdot t'$. By Property 4, this means that $o(\vec{t}[\sigma]) = x \cdot t'$. By Property 3, t' is either some $M(y)$ or $o'(\vec{u})$. But in the first case, $x \cdot t' = x \cdot M(y) = M(y \circ x)$ by Property 4, so it cannot equal $o(\vec{t}[\sigma])$, by Property 3. Therefore, $t' = o'(\vec{u})$ for some $o' : Kb \to O$ and $\vec{u} : \overline{o'} \to T\Delta$. By Property 4, $x \cdot t' = (x \cdot o')(\vec{u} \circ x^o)$. By Property 3, $o = x \cdot o'$, and $\vec{t}[\sigma] = \vec{u} \circ x^o$.

Remark 11. Note that if there were more than one possible o', then the most general unifier would not exist. But such a o', if it exists, is unique because Kx is epimorphic, by Property 1.(iii). In fact, this is the only place where we use this property. As a consequence, we could weaken the condition that morphisms in $\mathcal{A}$ are monomorphic and require instead that the image of such a morphism by O is monomorphic.

In case $o = x \cdot o'$, it follows from the above observations that a cocone is equivalently given by a cospan $T\Gamma \xrightarrow{\sigma} T\Delta \xleftarrow{\vec{u}} b^o$ such that $\vec{t}[\sigma] = \vec{u} \circ x^o$. But, by Remark 10, this is precisely the data for a pushout cocone for $\Gamma \xleftarrow{\vec{t}} a^o \xrightarrow{\mathcal{L}^+ x^o} b^o$. This actually induces an isomorphism between the two categories of cocones, thus justifying the rules P-Rig and P-Fail.

5.2 Flex

Here, we construct the pushout of $(\Gamma, M : c) \xleftarrow{M(y)} Ka \xrightarrow{N(x)} N : b$. Note that in this span, $N(x) = \mathcal{L}x$ while $M(y) = \mathcal{L}y[in_M]$. Thanks to the following lemma, it is actually enough to compute the pushout of $\mathcal{L}x$ and $\mathcal{L}y$.

Lemma 23. *In any category, denoting morphism composition by $f \circ g = g[f]$, the following rule applies*

$$\frac{X \vdash g :> f \Rightarrow u; \sigma \dashv Z}{X + Y \vdash g[in_1] :> f \Rightarrow u[in_1]; \sigma + Y \dashv Z + Y}$$

In other words, if the diagram below left is a pushout, then so is the right one.

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

By Property 1.(i) and Property 2, the pushout of $\mathcal{L}x$ and $\mathcal{L}y$ is the image by $\mathcal{L}$ of the pushout of x and y, thus justifying the rule P-Flex.

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 top-level of the l.h.s does not occur in the r.h.s. This section is devoted to the proof that if there is a unifier of $M(\vec{x})$ and t, then either M does not occur in t, or it occurs at top-level (see Corollary 3). The argument formalises the basic intuition that $t = u[M \mapsto t]$ is impossible if M occurs deeply in u because the sizes of both hand sides can never match. To make this statement precise, we need some recursive definitions and properties of size, formally justified by exploiting the universal property of TX as the free F-algebra on X.

Definition 3. *The size $|t| \in \mathbb{N}$ of a morphism $t : Ka \to T\Gamma$ is recursively defined by $|M(x)| = 0$ and $|o(\vec{t})| = 1 + |\vec{t}|$, with $|\vec{t}| = \sum_i t_i$, for any $\vec{t} : \coprod_i Ka_i \xrightarrow{\ldots,t_i,\ldots} T\Gamma$.*

We will also need to count the occurrences of a metavariables in a term.

Definition 4. *For each morphism $t : Ka \to T(\Gamma, M : b)$ we define $|t|_M$ recursively by $|M(x)|_M = 1$, $|N(x)|_M = 0$* if $N \neq M$, and $|o(\vec{t})|_M = |\vec{t}|_M$ *with the sum convention as above for $|\vec{t}|_M$.*

Remark 12. More formally, given $t : Ka \to T\Gamma$, the size $|t|$ is defined as the natural number n such that $1 \xrightarrow{n} \mathbb{N}$ factors $Ka \to T\Gamma \to \mathbb{N}$ (by Property 1.(v), since $\mathbb{N}$ is the coproduct $\coprod_{n\in\mathbb{N}} 1$), where $T\Gamma \to \mathbb{N}$ is the universal F-algebra morphism induced by the constant morphism $\Gamma \xrightarrow{0} \mathbb{N}$ and the F-algebra $F\mathbb{N} \cong \coprod_o \mathbb{N}^{J_o} \times S_o \xrightarrow{\coprod_o \pi_1} \coprod_o \mathbb{N}^{J_o} \xrightarrow{\coprod_o (1+\sum)} \coprod_o \mathbb{N} \to \mathbb{N}$, informally mapping $o(\vec{n})$ to $1 + \sum_j n_j$.

Given $t : Ka \to T(\Gamma, M : b)$, the natural number $|t|_M$ is computed similarly by the postcomposition with the universal F-algebra morphism $T(\Gamma, M : b) \to \mathbb{N}$ induced by $\Gamma, M : b \xrightarrow{0,1} \mathbb{N}$ and the F-algebra structure $F\mathbb{N} \cong \coprod_o \mathbb{N}^{J_o} \times S_o \xrightarrow{\coprod_o \pi_1} \coprod_o \mathbb{N}^{J_o} \xrightarrow{\coprod_o \sum} \coprod_o \mathbb{N} \to \mathbb{N}$, informally mapping $o(\vec{n})$ to $\sum_j n_j$.

The lemmas below are easy consequences of the following standard induction lemma.

Lemma 24. *Assume given, for each object a of $\mathcal{A}$, a predicate P_a on $\hom(Ka, T\Gamma)$ such that*

- *$P_a(M(x))$ holds for any $M : b \in \Gamma$ and $x \in \hom_{\mathscr{D}}(a, b)$;*
- *given $o \in O_a$ and $\vec{t} : \coprod_{j\in J_{a,o}} KL_{a,o,j} \to T\Gamma$, the property $P_a(o(\vec{t}))$ holds whenever $P_{L_{a,o,j}}(t_j)$ holds for every $j \in J_{a,o}$.*

Then, $P_a(t)$ holds for any $t : Ka \to T\Gamma$.

Proof. Consider the functor $X : \mathcal{A} \to \mathrm{Set}$ defined by $X_a = \{t \in Ka \to T\Gamma | \forall f : b \to a, P_a(f \cdot t)\}$. By the Yoneda lemma, there is an injective projection $X \to T\Gamma$. By universal property of $T\Gamma$ as the free F-algebra on Γ, this projection has a section, and is thus an isomorphism.

Lemma 25. *For any $t : Ka \to T(\Gamma, M : b)$, if $|t|_M = 0$, then $T\Gamma \hookrightarrow T(\Gamma, M : b)$ factors t. Moreover, for any $\Gamma = (M_1 : a_1, \ldots, M_n : a_n)$, $t : Ka \to T\Gamma$, and $\sigma : \Gamma \to T\Delta$, we have $|t[\sigma]| = |t| + \sum_i |t|_{M_i} \times |\sigma_i|$.*

Corollary 2. *For any $t : Ka \to T(\Gamma, M : b)$, $\sigma : \Gamma \to T\Delta$, $x \in \hom_{\mathscr{D}}(a, b)$, $u : Kb \to T\Delta$, we have $|t[\sigma, u]| \geq |t| + |u| \times |t|_M$ and $|M(x)[u]| = |u|$.*

Corollary 3. *If there is a commuting square in Kl_T*

$$\begin{tikzcd} Ka \arrow[r, "t"] \arrow[d, "M(x)"'] & \Gamma, M : b \arrow[d, "\sigma{,}u"] \\ M : b \arrow[r, "u"'] & \Delta \end{tikzcd}$$

then either $t = M(y)$ for some y, or $T\Gamma \hookrightarrow T(\Gamma, M : b)$ factors t.

Proof. Since $t[\sigma, u] = M(x)[u]$, we have $|t[\sigma, u]| = |M(x)[u]|$. Corollary 2 implies $|u| \geq |t| + |u| \times |t|_M$. Therefore, either $|t|_M = 0$ and we conclude by Lemma 25, or $|t|_M = 1$ and $|t| = 0$ and so t is $M(y)$ for some y.

7 Completeness

Each inductive rule presented so far provides an elementary step for the construction of coequalisers. We need to ensure that this set of rules allows to construct a coequaliser in a finite number of steps. To make the argument more straightforward, we explicitly assume that in the splitting rules U-Split and P-Split in figure 1, the expressions with vector notation are not empty lists.

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 given some input data, and *termination*, i.e., there is no infinite sequence of rule applications. In this section, we sketch the proof of the latter termination property, following a standard argument. 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 as its size as a finite family of elements of $\mathcal{A}$ (see Remark 5). We extend this definition to the case where $\Gamma = \bot$, by taking $|\bot| = 0$. We also define the size $||t||$ of a *term* $t : Ka \to T\Gamma$ as in Definition 3 except that we assign a size of 1 to metavariables, so that no term is of empty size.

Let us first quickly justify termination of the pruning phase. We define the size of a judgment $\Gamma \vdash f :> g \Rightarrow u; \sigma \dashv \Delta$ as $||f||$. It is straightforward to check that the sizes of the premises are strictly smaller than the size of the conclusion, for the two recursive rules P-Split and P-Rig of the pruning phase, thanks to the following lemmas

Lemma 26. *For any $t : Ka \to T\Gamma$ and $\sigma : \Gamma \to T\Delta$, if σ is a renaming, i.e., $\sigma = \mathcal{L}^+\sigma'$, for some σ', then $||t[\sigma]|| = ||t||$.*

Lemma 27. *If there is a finite derivation tree of $\Gamma \vdash f :> g \Rightarrow u; \sigma \dashv \Delta$ and $\Delta \neq \bot$, then $|\Gamma| = |\Delta|$ and σ is a* renaming*.*

Now, we tackle termination for the unification phase. We define the size of a judgment $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$ to be the pair $(|\Gamma|, ||t||)$. The following lemma ensures that for the two recursive rules U-Split and U-RigRig in the unification phase, the sizes of the premises are strictly smaller than the size of the conclusion, for the lexicographic order.

Lemma 28. *If there is a finite derivation tree of $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$, then $|\Gamma| \geq |\Delta|$, and moreover if $|\Gamma| = |\Delta|$ and $\Delta \neq \bot$, then σ is a renaming.*

8 Applications

In the examples, we motivate the definition of the category $\mathcal{A}$ based on what we expect from metavariable arities, following Remark 4. We also adopt the format of Remark 7, where the endofunctor F is defined as

$$F(X)_a \cong \coprod_{\ell \in V} \coprod_{o \in O_\ell} \prod_{j \in \gamma_\ell} X_{H_{\ell,j}(a,o)}$$

8.1 Simply-typed second-order syntax

In this section, we present the example of simply-typed λ-calculus. Our treatment generalises to any second-order binding signature (see [6]).

Let T denote the set of simple types generated by a set of simple types. A metavariable arity $\tau_1, \ldots, \tau_n \vdash \tau_f$ is given by a list of input types $\tau_1, \ldots, \tau_n$ and an output type τ_f. Substituting a metavariable $M : (\Gamma \vdash \tau)$ with another $M' : (\Gamma' \vdash \tau')$ requires that $\tau = \tau'$ and involves an injective renaming $\Gamma \to \Gamma'$. Thus, we consider $\mathcal{A} = \mathbb{F}_m[T] \times T$, where $\mathbb{F}_m[T]$ is the category of finite lists of elements of T and injective renamings between them.

The following table summarises the definition of the endofunctor F on $[\mathcal{A}, \mathrm{Set}]$ specifying the syntax, where $|\Gamma|_\tau$ denotes the number (as a cardinal set) of occurrences of τ in Γ.

Typing rule	$O_\ell(\Gamma \vdash \tau)$	$H_{\ell,j}(\Gamma \vdash \tau)$
$\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$	$\{v_i \| i \in \|\Gamma\|_\tau\}$	-
$\dfrac{\Gamma \vdash t : \tau' \Rightarrow \tau \quad \Gamma \vdash u : \tau'}{\Gamma \vdash t\ u : \tau}$	$\{a_{\tau'} \| \tau' \in T\}$	$H_{-,0} = \Gamma \vdash \tau' \Rightarrow \tau$ $H_{-,1} = \Gamma \vdash \tau'$
$\dfrac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x.t : \tau_1 \Rightarrow \tau_2}$	$\{l_{\tau_1,\tau_2} \| \tau = (\tau_1 \Rightarrow \tau_2)\}$	$H_{-,0} = \Gamma, \tau_1 \vdash \tau_2$

8.2 Arguments as sets

If we think of the arguments of a metavariable as specifying the available variables, then it makes sense to assemble them in a set rather than in a list. This motivates considering the category $\mathcal{A} = \mathbb{I}$ whose objects are natural numbers and a morphism $n \to p$ is a subset of $\{0, \ldots, p-1\}$ of cardinal n. For instance, $\mathbb{I}$ 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-calculus as in Section §1.2. Then, a metavariable takes as argument a set of available variables, rather than a list of distinct variables. In this approach, unifying two metavariables (see the rules U-FlexFlex and P-Flex) amount to computing a set intersection.

8.3 Arities as sets

In this example, we describe pure λ-calculus extended with metavariables whose arities are sets of free variables. They do not take any explicit argument and they cannot be applied to bound variables.

We adopt a locally nameless approach, with two kinds of variables: the named ones, chosen in an infinite set $\mathcal{V}$ of names (e.g., $\mathbb{N}$), and the unnamed ones, as before, which will be used for binding. We thus choose $\mathcal{A}$ to be $\mathbb{S} \times \mathbb{F}_m$ where $\mathbb{S}$ is the category of finite subsets of $\mathcal{V}$ and inclusions (not injections!) between them. Pure λ-calculus can be specified by an endofunctor F defined by $F(X)_{A,n} = n + A + X_{A,n+1} + X_{A,n} \times X_{A,n}$.

A metavariable arity, as an object of $\mathcal{A}$, consists of two components: a finite set of named variables, and a number of arguments among unnamed variables. Let us define pure arities as those whose second component is 0. A metavariable is said pure if its arity is, and a metavariable context is said pure if each metavariable is.

The pure metavariables are the ones mentioned at the beginning of this section. Unifying a pure metavariable with itself, as in the rule U-FlexFlex, is a no-op, while unifying a pure metavariable with another one (rule P-Flex) produces a new pure metavariable whose arity is the intersection of the input metavariable arities. Exploiting this observation, an easy induction shows that the most general unifier targets a pure metavariable context.

Lemma 29. *Assume an endofunctor for syntax as in Section §2.2. If $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$ or $\Gamma \vdash t :> \coprod_i \mathcal{L} f_i \Rightarrow u; \sigma \dashv \Delta$, and $\Delta \neq \bot$, then Δ is pure whenever Γ is.*

8.4 Quantum $\boldsymbol{\lambda}$-calculus

In this section we explain how we can define pattern unification for quantum λ-calculus [16]. We denote by S the set of types, which is inductively generated as follows

$$A, B, C \in S ::= \mathbf{qubit} | A \multimap B | !(A \multimap B) | 1 | A \otimes B | A + B | A^{\ell}$$

where A^{ℓ} is intuitively the type of finite lists of elements of type A.

We consider metavariable arities of the shape $\Delta \vdash A$, where Δ is the multiset of the argument types, and A is the output type of the metavariable. We denote by $!\Delta$ the non linear part of Δ, i.e., its sub-multiset consisting of its non-linear types, that is, types of the shape $!A$. We denote by $\boldsymbol{\Delta}$ the linear part of Δ. Substituting a metavariable of arity $\Delta_1 \vdash A_1$ with a metavariable of arity $\Delta_2 \vdash A_2$ requires that $A_1 = A_2$, $\boldsymbol{\Delta_1} = \boldsymbol{\Delta_2}$, and an injective renaming $!\Delta_1 \hookrightarrow !\Delta_2$. Therefore, we choose $\mathcal{A}$ to be the category whose objects are metavariable arities $\Delta \vdash A$ and the set of morphisms between $\Delta_1 \vdash A_1$ and $\Delta_2 \vdash A_2$ is empty if $A_1 \neq A_2$ or $\boldsymbol{\Delta_1} \neq \boldsymbol{\Delta_2}$, or is the set of injective renamings between $!\Delta_1$ and $!\Delta_2$ otherwise.

The components of the endofunctor F on $[\mathcal{A}, \mathrm{Set}]$ are specified in Table 1, except for the promotion, which we discuss below. We use the following notations.

Notation 12. *$\delta(P)$ denotes either a singleton set or the empty set, depending on whether the property P is true.*

$|\Delta|_{!C}$ denotes the number of occurrences of $!C$ in Δ. By convention, we take it to be 0 if $!C$ does not make sense (i.e., when C is not $A \multimap B$).

The first rule handles the term constants in $\mathcal{C} = \{\texttt{skip}, \texttt{split}^A, \texttt{meas}, \texttt{new}, U\}$, that all have typing rules of the following shape

$$\frac{c \in \mathcal{C}}{!\Delta \vdash c : A_c \multimap B_c}$$

Let us now discuss promotion for values.

$$\frac{!\Delta \vdash V : A \multimap B}{!\Delta \vdash V : !(A \multimap B)}\ p$$

This typing rule can be split as in the following table, depending on what V is: a variable, a λ-abstraction, or a term constant $c \in \mathcal{C} = \{\texttt{skip}, \texttt{split}^A, \texttt{meas}, \texttt{new}, U\}$.

Typing rules for values	$O_\ell(\Delta \vdash C)$	$H_{\ell,j}(\Delta \vdash C, o)$
$!\Delta, x : !(A \multimap B) \vdash x : !(A \multimap B)$	$\delta(\Delta = !\Delta) \times \lvert\Delta\rvert_C$	-
$\dfrac{!\Delta, x : A \vdash M : B}{!\Delta \vdash \lambda x^A.M : !(A \multimap B)}$	$\{l_{A,B,v} \lvert \Delta \vdash C = !\Delta \vdash !(A \multimap B)\}$	$H_{-,1}(l_{A,B,v}) = \Delta, A \vdash B$
$!\Delta \vdash c : !(A_c \multimap B_c)$	$\delta(\Delta \vdash C = !\Delta \vdash !(A_c \multimap B_c))$	-

8.5 Intrinsic polymorphic syntax

We present intrinsic system F, following [11]. Let $S : \mathbb{F}_m \to \mathrm{Set}$ mapping n to the set S_n of types for system F taking free type variables in $\{0, \ldots, n-1\}$. Intuitively, a metavariable arity $n; \sigma_1, \ldots, \sigma_p \vdash \tau$ specifies the number n of free type variables, the list of input types $\vec{\sigma}$, and the output type τ, all living in S_n. Substituting a metavariable $M : (n; \vec{\sigma} \vdash \tau)$ with another $M' : (n'; \vec{\sigma}' \vdash \tau')$ requires a choice $(\alpha_0, \ldots, \alpha_{n-1})$ of n distinct type variables among $\{0, \ldots n'-1\}$, such that $\tau[\vec{\alpha}] = \tau'$, and an injective renaming $\vec{\sigma'}[\vec{\alpha}] \to \vec{\sigma}$. We therefore consider the category $\mathcal{A}$ of metavariable arities where a morphism between $n; \Gamma \vdash \tau$ and $n'; \Gamma' \vdash \tau'$ is a morphism $\sigma : n \to n'$ in $\mathbb{F}_m$ such that $\tau[\sigma] = \tau'$, and a renaming $\Gamma[\sigma] \to \Gamma'$. More formally, $\mathcal{A}$ is the op-lax colimit of $n \mapsto \mathbb{F}_m[S_n] \times S_n$. The intrinsic syntax of system F can then be specified as follows.

Typing rules (Γ,Σ linear)	$O_\ell(\Delta \vdash C)$	$H_{\ell,j}(\Delta \vdash C, o)$
$\dfrac{c \in \mathcal{C}}{!\Delta \vdash c : A_c \multimap B_c}$	$\delta(\Delta \vdash C = !\Delta \vdash A_c \multimap B_c)$	-
$\dfrac{A \text{ linear}}{!\Delta, x : A \vdash x : A}\ ax$	$\delta(\boldsymbol{\Delta} = C)$	-
$\dfrac{}{!\Delta, x : !(A \multimap B) \vdash x : A \multimap B}\ axd$	$\delta(\Delta = !\Delta) \times \|\Delta\|_{!C}$	-
$\dfrac{\Delta, x : A \vdash M : B}{\Delta \vdash \lambda x^A.M : A \multimap B}\ \multimap_I$	$\{l_{A,B} \| C = A \multimap B\}$	$H_{-,1} = \Delta, A \vdash B$
$\dfrac{!\Delta, \Gamma \vdash M : A \multimap C \quad !\Delta, \Sigma \vdash N : A}{!\Delta, \Gamma, \Sigma \vdash MN : C}\ \multimap_E$	$\{a_{\Gamma,\Sigma,A} \| \Delta = !\Delta, \Gamma, \Sigma\}$	$H_{-,0} = !\Delta, \Gamma \vdash A \multimap C$ $H_{-,1} = !\Delta, \Sigma \vdash A$
$\dfrac{!\Delta, \Gamma \vdash M : 1 \quad !\Delta, \Sigma \vdash N : C}{!\Delta, \Gamma, \Sigma \vdash M;N : C}\ 1_E$	$\{u_{\Gamma,\Sigma} \| \Delta = !\Delta, \Gamma, \Sigma\}$	$H_{-,0} = !\Delta, \Gamma \vdash 1$ $H_{-,1} = !\Delta, \Sigma \vdash C$
$\dfrac{!\Delta, \Gamma \vdash M : A \quad !\Delta, \Sigma \vdash N : B}{!\Delta, \Gamma, \Sigma \vdash M \otimes N : A \otimes B}\ \otimes_I$	$\{t_{A,B,\Gamma,\Sigma} \| \Delta \vdash C = !\Delta, \Gamma, \Sigma \vdash A \otimes B\}$	$H_{-,0} = !\Delta, \Gamma \vdash A$ $H_{-,1} = !\Delta, \Sigma \vdash B$
$\dfrac{!\Delta, \Gamma \vdash M : A \otimes B \quad !\Delta, \Sigma, x : A, y : B \vdash N : C}{!\Delta, \Gamma, \Sigma \vdash \texttt{let}\ x^A \otimes y^B\ =\ M\ \texttt{in}\ N : C}\ \otimes_E$	$\{t'_{A,B,\Gamma,\Sigma} \| \Delta = !\Delta, \Gamma, \Sigma\}$	$H_{-,0} = !\Delta, \Gamma \vdash A \otimes B$ $H_{-,1} = !\Delta, \Sigma, A, B \vdash C$
$\dfrac{!\Delta, \Gamma \vdash M : A}{!\Delta, \Gamma \vdash \texttt{in}_\ell\ M : A \oplus B}\ \oplus_I^\ell$	$\{inl_{A,B} \| C = A \oplus B\}$	$H_{-,0} = \Delta \vdash A$
$\dfrac{!\Delta, \Gamma \vdash M : B}{!\Delta, \Gamma \vdash \texttt{in}_r\ M : A \oplus B}\ \oplus_I^r$	$\{inr_{A,B} \| C = A \oplus B\}$	$H_{-,0} = \Delta \vdash B$
$\dfrac{\begin{matrix} & !\Delta, \Sigma, x : A \vdash M : C \\ !\Delta, \Gamma \vdash P : A \oplus B & !\Delta, \Sigma, y : B \vdash N : C\end{matrix}}{!\Delta, \Gamma, \Sigma \vdash \texttt{match}\ P\ \texttt{with}\ (x^A : M \mid y^B : N) : C}\ \oplus_E$	$\{m_{A,B,\Gamma,\Sigma} \| \Delta =!\Delta, \Gamma, \Sigma\}$	$H_{-,0} = !\Delta, \Gamma \vdash A \oplus B$ $H_{-,1} = !\Delta, \Sigma, A \vdash C$ $H_{-,2} = !\Delta, \Sigma, B \vdash C$
$\dfrac{!\Delta, \Gamma \vdash M : 1 \oplus (A \otimes A^\ell)}{!\Delta, \Gamma \vdash M : A^\ell}\ -_I^\ell$	$\{tail_A \| C = A^\ell\}$	$H_{-,0} = \Delta \vdash 1 \oplus (A \otimes A^\ell)$
$\dfrac{\begin{matrix} !\Delta, \Gamma, f : !(A \multimap B) \vdash N : C \\ !\Delta, f : !(A \multimap B), x : A \vdash M : B\end{matrix}}{!\Delta, \Gamma \vdash \texttt{letrec}\ f^{A \multimap B}\, x = M\ \texttt{in}\ N : C}\ \texttt{rec}$	$\{rec_{A,B} \| A, B \in S\}$	$H_{-,0} = \Delta, f : !(A \multimap B) \vdash C$ $H_{-,1} = !\Delta, f : !(A \multimap B), A \vdash B$

Table 1. Some components of the endofunctor specifying the quantum λ-calculus.

Typing rule	$O_\ell(n; \Gamma \vdash \tau)$	$H_{\ell,j}(n; \Gamma \vdash \tau, o)$
$\dfrac{x : \tau \in \Gamma}{n; \Gamma \vdash x : \tau}$	$\{v_i \| i \in \|\Gamma\|_\tau\}$	-
$\dfrac{n; \Gamma \vdash t : \tau' \Rightarrow \tau \quad n; \Gamma \vdash u : \tau'}{n; \Gamma \vdash t\ u : \tau}$	$\{a_{\tau'} \| \tau' \in S_n\}$	$H_{-,0} = n; \Gamma \vdash \tau' \Rightarrow \tau$ $H_{-,1} = n; \Gamma \vdash \tau'$
$\dfrac{n; \Gamma, x : \tau_1 \vdash t : \tau_2}{n; \Gamma \vdash \lambda x.t : \tau_1 \Rightarrow \tau_2}$	$\{l_{\tau_1,\tau_2} \| \tau = (\tau_1 \Rightarrow \tau_2)\}$	$H_{-,0} = n; \Gamma, \tau_1 \vdash \tau_2$
$\dfrac{n; \Gamma \vdash t : \forall \tau_1 \quad \tau_2 \in S_n}{n; \Gamma \vdash t \cdot \tau_2 : \tau_1[\tau_2]}$	$\{A_{\tau_1,\tau_2} \| \tau = \tau_1[\tau_2]\}$	$H_{-,0} = n; \Gamma \vdash \forall \tau_1$
$\dfrac{n+1; wk(\Gamma) \vdash t : \tau}{n; \Gamma \vdash \Lambda t : \forall \tau}$	$\{\Lambda_{\tau'} \| \tau = \forall \tau'\}$	$H_{-,0} = n+1; wk(\Gamma) \vdash \tau'$

To understand how unification of two metavariables works (see the rules U-RigRig and P-Rig), let us explain how finite connected limits are computed in $\mathcal{A}$.

Let us introduce the category $\mathcal{A}'$ whose definition follows that of $\mathcal{A}$, but without the output types: objects are pairs of a natural number n and an element of S_n. Note that this is op-lax colimit of $n \mapsto \mathbb{F}_m[S_n]$, and there is an obvious projection $\mathcal{A} \to \mathcal{A}'$, which creates finite limits, as we will show.

Lemma 30. *$\mathcal{A}'$ has finite limits, and the projection functor $\mathcal{A}' \to \mathbb{F}_m$ preserves them.*

Proof. The crucial point is that $\mathcal{A}'$ is not only op-fibred over $\mathbb{F}_m$ by construction, it is also fibred over $\mathbb{F}_m$. Intuitively, if $\Gamma \in \mathbb{F}_m[S_n]$ and $f : n' \to n$ is a morphism in $\mathbb{F}_m$, then $f_! \Gamma \in \mathbb{F}_m[S_{n'}]$ is essentially Γ restricted to elements of S_n that are in the image of S_f. Note that $f_!$ is right adjoint to $\Gamma \mapsto \Gamma[f]$, and is thus continuous. We now apply [10, Theorem 4.2 and Proposition 4.1]: each $\mathbb{F}_m[S_n]$ has those limits.

Lemma 31. *The projection functor $\mathcal{A} \to \mathcal{A}'$ creates finite limits.*

Proof. Let $d : I \to \mathcal{A}$ be a functor. We denote d_i by $n_i; \Gamma_i \vdash \tau_i$. Let $n; \Gamma$ be the limit of $i \mapsto n_i; \Gamma_i$ in $\mathcal{A}'$. By the previous lemma, n is the limit of $i \mapsto n_i$. Note that $S : \mathbb{F}_m \to \mathrm{Set}$ preserves finite connected limits. Thus, we can define $\tau \in S_n$ as corresponding to the universal function $1 \to S_n$ factorising the cone $(1 \xrightarrow{\tau_i} S_{n_i})_i$.

It is easy to check that $n; \Gamma \vdash \tau$ is the limit of d.

More concretely, a finite connected limit of $i \mapsto n_i; \Gamma_i \vdash \tau_i$ in $\mathcal{A}$ is computed as follows:

1. compute the limit n of $(n_i)_i$ in $\mathbb{F}_m$, denoting $p_i : n \to n_i$ is the canonical projections;
2. define τ as the (only) element of S_n such that $\tau[p_i] = \tau_i$
3. define Γ as the limit in $\mathbb{F}_m[S_n]$ of $p_{i,!}\Gamma_i$, where $p_{i,!} : \mathbb{F}_m[S_{n_i}] \to \mathbb{F}_m[S_n]$ is the reindexing functor described in the proof of Lemma 30.

This means that to unify $M(\vec{\alpha};\vec{\tau})$ with $M(\vec{\alpha}';\vec{\tau}')$, with M of arity $p;\vec{u} \vdash u'$, we first need to compute the vector of common position $\vec{i}$ between $\vec{\alpha}$ and $\vec{\alpha}'$, i.e, the largest vector $(i_1 < \cdots < i_n)$ such that $\alpha_{i_j} = \alpha'_{i_j}$. Then, we consider $\vec{\sigma}$ and $\vec{\sigma}'$ so that $\vec{\sigma}[j \mapsto i_{j+1}]$ is the sub-list of $\vec{\tau}$ that only use type variables in $\alpha_{i_1}, \ldots, \alpha_{i_n}$, and similarly for $\vec{\sigma}'$ and $\vec{\tau}'$. Finally, we define $(t_1, \ldots, t_m)$ as the vector of common positions between $\vec{\sigma}$ and $\vec{\sigma}'$. The most general unifier is then $M \mapsto N(\vec{i}, \vec{t})$ for a fresh metavariable N of arity $n; \sigma_{t_1}, \ldots \sigma_{t_m} \vdash t'$, where t' is $u'[i_{j+1} \mapsto j]$.

References

1. Adámek, J., Rosicky, J.: Locally Presentable and Accessible Categories. Cambridge University Press (1994). `https://doi.org/10.1017/CBO9780511600579`
2. Adámek, J., Borceux, F., Lack, S., Rosicky, J.: A classification of accessible categories. Journal of Pure and Applied Algebra **175**(1), 7–30 (2002). `https://doi.org/https://doi.org/10.1016/S0022-4049(02)00126-3`, `https://www.sciencedirect.com/science/article/pii/S0022404902001263`, special Volume celebrating the 70th birthday of Professor Max Kelly
3. Borceux, F.: Handbook of Categorical Algebra 1: Basic Category Theory. Encyclopedia of Mathematics and its Applications, Cambridge University Press (1994). `https://doi.org/10.1017/CBO9780511525858`
4. D., P.G.: A note on inductive generalization. Machine Intelligence **5**, 153–163 (1970), `https://cir.nii.ac.jp/crid/1573387448853680384`
5. De Bruijn, N.G.: Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. Indagationes Mathematicae **34**, 381–392 (1972)
6. Fiore, M.P., Hur, C.K.: Second-order equational logic. In: Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010) (2010)
7. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax involving binders. In: Proc. 14th Symposium on Logic in Computer Science IEEE (1999)
8. Goguen, J.A.: What is unification? - a categorical view of substitution, equation and solution. In: Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques. pp. 217–261. Academic (1989)
9. Goldfarb, W.D.: The undecidability of the second-order unification problem. Theoretical Computer Science **13**(2), 225–230 (1981). `https://doi.org/https://doi.org/10.1016/0304-3975(81)90040-2`, `https://www.sciencedirect.com/science/article/pii/0304397581900402`
10. Gray, J.W.: Fibred and cofibred categories. In: Eilenberg, S., Harrison, D.K., MacLane, S., Röhrl, H. (eds.) Proceedings of the Conference on Categorical Algebra. pp. 21–83. Springer Berlin Heidelberg, Berlin, Heidelberg (1966)
11. Hamana, M.: Polymorphic abstract syntax via grothendieck construction (2011)

12. Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. **1**(1), 27–57 (1975). https://doi.org/10.1016/0304-3975(75)90011-0, https://doi.org/10.1016/0304-3975(75)90011-0
13. Lafont, A., Krishnaswami, N.: Generic pattern unification: a categorical approach (2022), https://raw.githubusercontent.com/amblafont/unification/master/lncs-long.pdf, preprint
14. Mac Lane, S.: Categories for the Working Mathematician. No. 5 in Graduate Texts in Mathematics, Springer, 2nd edn. (1998)
15. Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. J. Log. Comput. **1**(4), 497–536 (1991). https://doi.org/10.1093/logcom/1.4.497, https://doi.org/10.1093/logcom/1.4.497
16. Pagani, M., Selinger, P., Valiron, B.: Applying quantitative semantics to higher-order quantum computing. In: Jagannathan, S., Sewell, P. (eds.) The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014. pp. 647–658. ACM (2014). https://doi.org/10.1145/2535838.2535879, https://doi.org/10.1145/2535838.2535879
17. Plotkin, G.: An illative theory of relations. In: Cooper, R., et al. (eds.) Situation Theory and its Applications. pp. 133–146. No. 22 in CSLI Lecture Notes, Stanford University (1990)
18. Reiterman, J.: A left adjoint construction related to free triples. Journal of Pure and Applied Algebra **10**, 57–71 (1977)
19. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (jan 1965). https://doi.org/10.1145/321250.321253, https://doi.org/10.1145/321250.321253
20. Rydeheard, D.E., Burstall, R.M.: Computational category theory. Prentice Hall International Series in Computer Science, Prentice Hall (1988)
21. Vezzosi, A., Abel, A.: A categorical perspective on pattern unification. RISC-Linz p. 69 (2014)