Generic pattern unification: a categorical approach

Ambroise Lafont[0000−0002−9299−641X]

University of Cambridge

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

Keywords: Unification · Category theory.

1 Introduction

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

1.1 A short introduction to pattern unification

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

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

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

1.2 Generic pattern unification

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

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

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

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

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

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

to the substituted term $t[\sigma]$. Now, a unification problem can be stated as pair of parallel Kleisli morphisms

$$yp \rightrightarrows T(yq_1 + \cdots + yq_n)$$

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

Related work

First-order unification was categorically described in [2]. Pattern unification was introduced in [4], as a particular case of higher-order unification for the simply-typed lambda-calculus, where metavariables are applied to distinct variables. It was categorically rephrased in [6], with concluding hints about how to generalise their work. The present paper can be viewed as an explicit realisation of this generalisation.

Plan of the paper

In Section 3, we state our main result that justifies pattern unification algorithms. Then we tackle the proof algorithm summarised in Figure 1.2, starting with the the unification phase (Section 4), the pruning phase (Section 5), , the occur-check (Section 6), and finally we justify completeness (Section 7). Applications are presented in Section 8.

General notations

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

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

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

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

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

Given an adjunction $L \dashv R$ and a morphism $f : A \to RB$, we denote by $f^* : LA \to B$ its transpose, and similarly, if $g : LA \to B$, then $g^* : A \to RB$.

Unification phase

– Structural rules

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

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

– Rigid-rigid

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

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

– Flex-*

$$\frac{\begin{array}{ccc} Ka & \xrightarrow{u} & T(\Gamma, M:b) \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle m_{\Gamma;b}} \\ T\Gamma & \xrightarrow[in_1]{} & T\Gamma + 1 \end{array} \qquad \Gamma \vdash g :> f \Rightarrow v; \sigma \dashv \Delta}{\Gamma, M:b \vdash M(f) = u \Rightarrow \sigma, M \mapsto v \dashv \Delta} \quad + \text{symmetric rule}$$

$$\frac{\begin{array}{ccc} a & \xrightarrow{u} & T(\Gamma, M:b) \\ \downarrow & & \downarrow{\scriptstyle m_{\Gamma;b}} \\ 1 & \xrightarrow[in_2]{} & T\Gamma + 1 \end{array} \qquad u = o(g;s)}{\Gamma, M:b \vdash M(f) = u \Rightarrow ! \dashv \top} \quad + \text{symmetric rule}$$

$$\frac{Ka \underset{g}{\overset{f}{\rightrightarrows}} b - \overset{h}{-} \succ c \ \text{ is a coequaliser in } \mathscr{D} = \mathcal{A}^{op}}{\Gamma, M:b \vdash M(f) = M(g) \Rightarrow M \mapsto M'(h) \dashv \Gamma, M':c}$$

Pruning phase

– Structural rules

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

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

– Rigid

$$\frac{Kf : Ka \to Kb \text{ does not factor } s : Ka \to S_o}{\Gamma \vdash o(g;s) :> f \Rightarrow !; ! \dashv \top} \qquad \frac{\Gamma \vdash g :> f^o \Rightarrow u; \sigma \dashv \Delta \qquad s = s' \circ Kf}{\Gamma \vdash o(g;s) :> f \Rightarrow o(u;s'); \sigma \dashv \Delta}$$

– Flex

$$\frac{\begin{array}{ccc} a & \xrightarrow{f} & b \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle g'} \\ c & \xrightarrow[f']{} & d \end{array} \ \text{ is a pushout in } \mathscr{D} = \mathcal{A}^{op}}{\Gamma, M:c \vdash M(g) :> f \Rightarrow M'(g'); M \mapsto M'(f') \dashv \Gamma, M':d}$$

Fig. 1. Summary of the rules

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 $\mathscr{B}$. Any Kleisli morphism $f : A \to TB$ induces a morphism $f^* : TA \to TB$. We denote the Kleisli composition of $f : A \to TB$ and $g : TB \to T\Gamma$ by $f[g] = g^* \circ f$. We denote by $\mathcal{L}$ the left adjoint $\mathcal{L} : \mathscr{B} \to Kl_T$ which is the identity on objects and postcomposes any morphism $A \to B$ by $\eta_B : B \to TB$.

2 General setting

2.1 Base category

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

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

Remark 1. An object of $\mathcal{A}$ is intuitively a metavariable arity, or a ground context. The argument of a metavariable $M : a$ living in a ground context a' is a morphism $a \to a'$. In the previous example, a metavariable $M : n$ living in a context $m \in \mathbb{N}$ takes as argument a choice of distinct variables $x_1, \ldots, x_n \in \{0, \ldots, m-1\}$.

Remark 2. The restriction to functors preserving finite connected limits will justify that the case where we unify two metavariables: the result is then a new metavariable, whose arity is computed in $\mathcal{A}$.

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

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

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

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

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

Remark 3. We need those cocompleteness properties so that we can compute free monads of a finitary endofunctor as the colimit of an initial chain.

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

Remark 4. $\mathscr{D}^+$ is equivalent to the category of finite families of objects of $\mathcal{A}$. Since objects of $\mathcal{A}$ are intuitvely metavariable arities (Remark 1), $\mathscr{D}^+$ can be thought of as the category of metavariable contexts.

In the spirit of this remark, we introduce the usual context notation for objects of $\mathscr{D}^+$.

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

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

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

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

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

The following two properties are direct consequences of Lemma 1.

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

2.2 The endofunctor for syntax

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

$$F(X) = \coprod_{o \in O} \prod_{j \in J_o} X \circ L_{o,j} \times S_o,$$

for some set O, where, for each $o \in O$,

- $S_o \in \mathscr{C}$;
- J_o is a finite set;
- for each $j \in J_o$, $L_{o,j}$ is an endofunctor on $\mathcal{A}$ preserving finite connected limits.

Remark 5. S_o is, for instance, the variable presheaf (in this case, J_o is empty) or a type flag (in the simply-typed case).

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

We will rely on the following result.

Lemma 2. *F is finitary and restricts as an endofunctor on $\mathscr{C}$.*

Corollary 1. *F generates a free monad T that restricts to a monad on $\mathscr{C}$. Moreover, TX is the initial algebra of $Z \mapsto X + FZ$, as an endofunctor on $[\mathcal{A}, \mathrm{Set}]$ (or on $\mathscr{C}$, if X is in $\mathscr{C}$).*

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

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

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

Notation 24. *Given an index $o \in O$, a morphism $s : Ka \to S_o$, and $f : a^o \to TX$, we denote the induced morphism*

$$Ka \to FTX \hookrightarrow TX$$

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

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

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

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

by $M_i(f)$.

Property 8. Let $\Gamma = M_1 : a_1, \ldots, M_n : a_n \in \mathscr{D}^+$. Then, any morphism $u : Ka \to T\Gamma$ is one of the two mutually exclusive following possibilities:

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

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

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

Moreover, for any $u : b \to a$,

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

We end this section by introducing notations for Kleisli morphisms.

Notation 25. *Let Γ and Δ be contexts and $a \in \mathscr{D}$. Any $t : Ka \to T(\Gamma + \Delta)$ induces a Kleisli morphism $\Gamma, M : a \to T(\Gamma + \Delta)$ that we denote by $M \mapsto t$.*

3 Main result

The main point of pattern unification is that a coequaliser diagram in Kl_T selecting objects in $\mathscr{D}^+$ either has no unifier, either has a colimiting cocone. Working this logical disjunction is slightly inconvenient; we rephrase it in terms of a true coequaliser by freely adding a terminal object.

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

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

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

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

This lemma allows us to work with true coequalisers in Kl_T^*. The following result is also useful.

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

This has the following useful consequences:

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

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

Here is our main result.

Theorem 1. *Let $Kl^*_{\mathscr{D}^+}$ be the full subcategory of Kl_T^* consisting of objects of $\mathscr{D}^+ \cup \{\top\}$. Then, $Kl^*_{\mathscr{D}^+}$ has coequalisers.*

In other words, for any coequaliser diagram $A \rightrightarrows TB$ in $Kl_{\mathscr{D}^+}$ where A and B are in $\mathscr{D}^+$, either there is no cocone, either there is a coequaliser $B \to TC$, with $C \in \mathscr{D}^+$.

4 Unification phase

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

$$\coprod_i Ka_i \overset{t}{\underset{u}{\rightrightarrows}} \Gamma - \overset{\sigma}{-} \succ \Delta$$

We denote such a situation by

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

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

Let us start with simple cases:

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

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

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

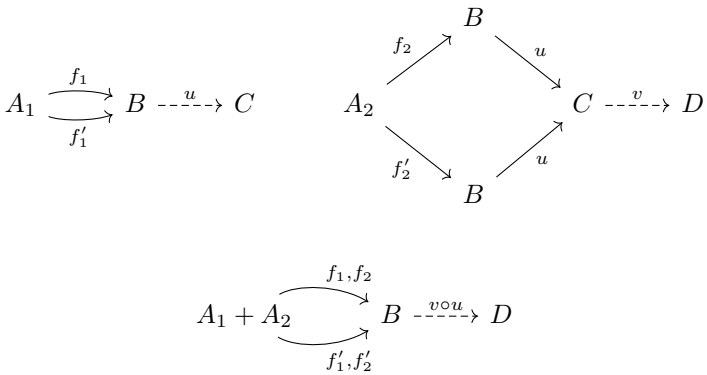
$$\frac{}{\Gamma \vdash () = () \Rightarrow 1_\Gamma \dashv \Gamma}.$$

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

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

This is justified by a general stepwise construction of coequalisers:

Lemma 6. *In any category, if the first two diagrams below are coequalisers, then the last one as well*



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

$$Ka \underset{u}{\overset{t}{\rightrightarrows}} T\Gamma$$

By Lemma 8, $t, u : Ka \to T\Gamma$ are one of the two following possibilites:

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

In the next subsections, we discuss the different possibilities.

4.1 Rigid-rigid

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

Assume given a unifier, i.e., a Kleisli morphism $\sigma : \Gamma \to T\Delta$ such that $t[\sigma] = u[\sigma]$. By Lemma 3, this means that $o(f[\sigma]; s) = o'(f'[\sigma]; s')$. By Lemma 8, this implies that $o = o'$, $f[\sigma] = f'[\sigma]$, and $s = s'$.

Therefore, we get the following failing rules

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

We now assume $o = o'$ and $s = s'$. Then, σ unifies t and u if and only if it unifies f and f'. This induces an isomorphism between the category of unifiers (i.e., cocones) for t and u and the category of unifiers for f and g. We therefore get the successful rule

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

4.2 Flex-*

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

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

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

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

We investigate the other cases in the subsections below, but first, let us make this rule more effective.

Lemma 7. *There is a morphism $m_{\Gamma;b} : T(\Gamma, M : b) \to T\Gamma + 1$ such that the following square commutes and is a pullback.*

$$\begin{array}{ccc} T\Gamma & \xrightarrow{T in_\Gamma} & T(\Gamma, M : b) \\ \| & & \downarrow{\scriptstyle m_{\Gamma;b}} \\ T\Gamma & \xrightarrow[in_1]{} & T\Gamma + 1 \end{array}$$

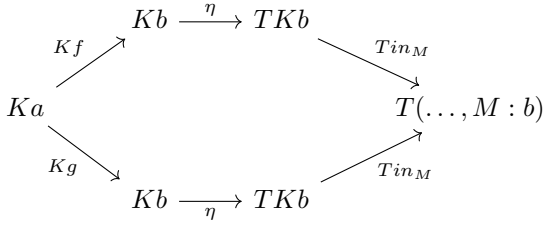
Proof. The proof consists in equipping $T\Gamma + 1$ with an adequate F-algebra considering the embedding $\Gamma, M : b \xrightarrow{\eta+!} T\Gamma + 1$. We then get the desired morphism by universal property of $T(\Gamma, M : b)$.

Corollary 2. *A morphism $u : Ka \to T(\Gamma, M : b)$ factors through $T\Gamma \hookrightarrow T(\Gamma, M : b)$ if and only if it factors through $in_1 : T\Gamma \to T\Gamma + 1$ by postcomposing it with $m_{\Gamma;b}$. Otherwise, u factors through $in_2 : 1 \to T\Gamma + 1$.*

Therefore, we rephrase the above rule as follows.

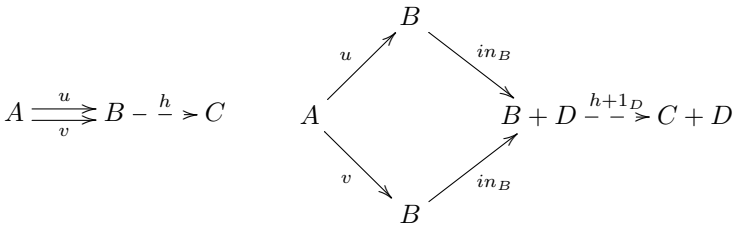
$$\frac{\begin{array}{ccc} a & \xrightarrow{u} & T(\Gamma', M : b) \\ \downarrow & & \downarrow{\scriptstyle m_{\Gamma';b}} \\ 1 & \xrightarrow[in_2]{} & T\Gamma' + 1 \end{array} \qquad u = o(g; s)}{\Gamma', M : b \vdash M(f) = u \Rightarrow ! \dashv \top}$$

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



We exploit the following lemma in Kl_T, with $u = \mathcal{L}Kf$ and $v = \mathcal{L}Kg$.

Lemma 8. *In any category, if the below left diagram is a coequaliser, then so is the below right diagram.*



Therefore, it is enough to compute the coequaliser of $\mathcal{L}Kf$ and $\mathcal{L}Kg$. Since $\mathcal{L}$ is left adjoint (and thus preserves coequalisers) and K preserves coequalisers (Property 3), we finally get the rule

$$\frac{Ka \underset{g}{\overset{f}{\rightrightarrows}} b - \overset{h}{-} \succ c \ \text{ is a coequaliser in } \mathscr{D} = \mathcal{A}^{op}}{\Gamma, M : b \vdash M(f) = M(g) \Rightarrow M \mapsto M'(h) \dashv \Gamma, M' : c}$$

Note that such a coequaliser always exists by Property 3.

Remark 6. If we use the same notation for coequalisers in $\mathscr{D}$, then this rule becomes

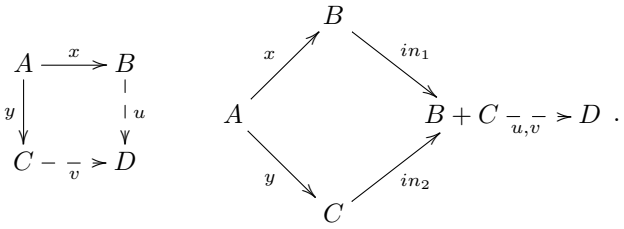
$$\frac{b \vdash f = g \Rightarrow h \dashv c}{\Gamma, M : b \vdash M(f) = M(g) \Rightarrow M \mapsto M'(h) \dashv \Gamma, M' : c}$$

4.3 Flex-rigid (successful occur-check)

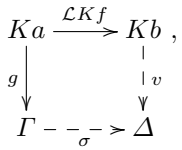
In this section, we consider a coequaliser diagram $Ka \overset{t}{\underset{u}{\rightrightarrows}} T\Gamma$ such that $\Gamma = \Gamma', M : b$ and $t = M(f)$ for some $f : a \to b$, and u factors as $Ka \xrightarrow{g} T\Gamma' \hookrightarrow T\Gamma$.

We exploit the following general lemma with $x = \mathcal{L}Kf$ and $y = g$.

Lemma 9. *In any category, if the below left diagram is a pushout, then the below right diagram is a coequaliser.*



Therefore, it is enough to compute the pushout



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

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

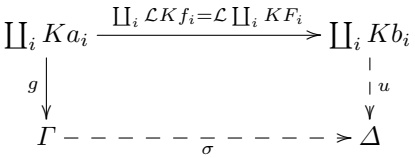
which we rephrase using Corollary (2) as

$$\frac{\begin{array}{ccc} Ka & \xrightarrow{u} & T(\Gamma, M : b) \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle m_{\Gamma;b}} \\ T\Gamma & \xrightarrow[in_1]{} & T\Gamma + 1 \end{array} \qquad \Gamma \vdash g :> f \Rightarrow v; \sigma \dashv \Delta}{\Gamma, M : b \vdash M(f) = u \Rightarrow \sigma, M \mapsto v \dashv \Delta}$$

5 Pruning phase

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

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



We denote such a situation by

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

Remark 7. For the intuition behind the notation and the relation to the so-called pruning process, consider the case of λ-calculus, in the category Nom of nominal sets. A span $\Gamma \xleftarrow{g} Kn \xrightarrow{\mathcal{L}f} Km$ corresponds to a term in $t \in T\Gamma_n$ and a choice of distinct m variables in $\{0, \ldots, n-1\}$, that is, an injection $f : m \to n$. Then, the pushout, if it exists, consists in "coercing" (hence the symbol :>) the term t to live in $T\Gamma_m$, by restricting the arity of the metavariables, as $\sigma : \Gamma \to T\Delta$ does. The resulting term $u \in \hom(Kn, T\Delta) \cong T\Delta_n$ is t but living in the "smaller" context $\{0, \ldots, m-1\}$ and with the restricted metavariables.

Remark 8. A cocone consists in morphisms $\coprod_i Kb_i \xrightarrow{t} T\Delta \xleftarrow{\sigma} \Gamma$ such that $g[\sigma] = t \circ f^o$, i.e., for all $i \in I$, we have $g_i[\sigma] = t_i \circ Kf$. Note that given σ, there is at most one t that works because Kf is epimorphic.

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

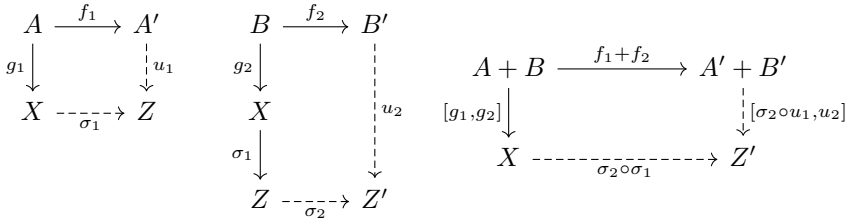
$$\frac{}{\Gamma \vdash () :> () \Rightarrow (); 1_\Gamma \dashv \Gamma}$$

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

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

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

Lemma 10. *In any category, if the first two diagrams below are coequalisers, then the last one as well*



Thus we have the following rule.

$$\frac{\Gamma \vdash g_1 :> f_1 \Rightarrow u_1; \sigma_1 \dashv \Delta_1 \qquad \Delta_1 \vdash g_2 :> f_2[\sigma_1] \Rightarrow u_2; \sigma_2 \dashv \Delta_2}{\Gamma \vdash g_1, g_2 :> f_1 + f_2 \Rightarrow u_1[\sigma_2], u_2; \sigma_1[\sigma_2] \dashv \Delta_2} \text{P-SPLIT} \quad (3)$$

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

$$\begin{array}{ccc} Ka & \xrightarrow{\mathcal{L}Kf} & Kb \\ \downarrow & & \\ \Gamma & & \end{array}$$

By Lemma 8, the left vertical morphism $Ka \to T\Gamma$ is one of the two following possibilites, investigated separately in the following subsections:s

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

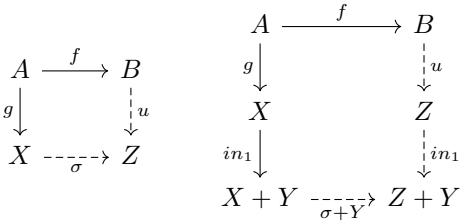
5.1 Flex

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

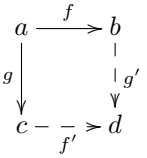
$$\begin{array}{ccc} Ka & \xrightarrow{\mathcal{L}Kf} & Kb \\ {\scriptstyle \mathcal{L}Kg}\downarrow & & \\ Kc & & \\ {\scriptstyle in_M}\downarrow & & \\ \Gamma, M : c & & \end{array}$$

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

Lemma 11. *In any category, if the diagram below left is a pushout, then so is the right one.*



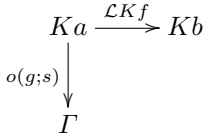
Since $\mathcal{L}$ is left adjoint (and thus preserves coequalisers) and K preserves pushouts (Property 3), the pushout can be computed in $\mathscr{D}$. Therefore, we get the rule



$$\frac{\text{is a pushout in } \mathscr{D} = \mathcal{A}^{op}}{\Gamma, M : c \vdash M(g) :> f \Rightarrow M'(g'); M \mapsto M'(f') \dashv \Gamma, M' : d}$$

5.2 Rigid

We want to compute the pushout



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

By Remark 8, a cocone in Kl_T is given by an object Δ with morphisms $Kb \xrightarrow{u} T\Delta \xleftarrow{\sigma} \Gamma$ such that $o(g; s)[\sigma] = u \circ Kf$. By Lemma 3, this means that $o(g[\sigma]; s) = u \circ Kf$. Now, u is either some $M'(f')$ or $o'(g'; s')$, with $g' : b^o \to T\Delta$ and $s' : Kb \to S_{o'}$. But in the first case, $u \circ Kf = M'(f') \circ Kf = M'(f' \circ f)$ so it cannot equal $o(g[\sigma]; s)$. So we are in the second case, and for the same reason, $o = o'$, $g[\sigma] = g' \circ f^o$ and $s = s' \circ Kf$. Note that such a s' is unique because Kf is epimorphic. Therefore we get the rule

$$\frac{Kf : Ka \to Kb \text{ does not factor } s : Ka \to S_o}{\Gamma \vdash o(g; s) :> f \Rightarrow !; ! \dashv \top} \tag{4}$$

Otherwise we get the following rule

$$\frac{\Gamma \vdash g :> f^o \Rightarrow u; \sigma \dashv \Delta \qquad s = s' \circ Kf}{\Gamma \vdash o(g; s) :> f \Rightarrow o(u; s'); \sigma \dashv \Delta} \tag{5}$$

6 Occur-check

The occur-check allows to jump from the main unification phase (Section 4) to the pruning phase (Section 5), whenever the metavariable appearing at the toplevel of the l.h.s does not appear in the r.h.s. This section is devoted to the proof that if there is a unifier, then the metavariable does not appear on the r.h.s, either it appears at top-level (see Corollary 3).

We prove it by working in the category $[\mathcal{A}, \mathrm{Set}]$.

Proposition 1. *Let R be any free monad on $[\mathcal{A}, \mathrm{Set}]$ generated by an endofunctor G of the shape as described in Section 2.2. Let $A \xrightarrow{n} R\emptyset$ be a natural transformation morphism, where A is representable. Then, the following diagram is a pullback*

$$\begin{array}{ccc} A + A \times_{R\emptyset} A & \longrightarrow & RA \\ \downarrow & & \downarrow{\scriptstyle n^*} \\ A & \xrightarrow[n]{} & R\emptyset \end{array}$$

Example 3. Taking $G = F$, $R = T$, and $A = (M : a)$, the pullback property means that given $f \in \hom_{\mathscr{D}}(a', a)$, and $t : Ka' \to TKa$ such that $t[n] = n \circ Kf$, either $t = M(g)$ for some g, either $t = n \circ Kf$. In particular, if $f = 1_a$, then $t[n] = n$ implies that $t = M(g)$ for some g or $t = n$.

Corollary 3. *Given $\sigma : \Gamma \to T1$, $u : Kd \to T1$, $f : Kc \to Kd$, if the following square commutes, then the top morphism factors through either $T\Gamma \to T(Kd+\Gamma)$ or $Kd \to TKd$*

$$\begin{array}{ccc} Kc & \longrightarrow & T(Kd + \Gamma) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle [u,\sigma]^*} \\ Kd & \xrightarrow[u]{} & T1 \end{array}$$

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

7 Completeness

Each inductive rule presented so far provides an elementary step for the construction of coequalisers. 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 slightly alter the splitting rules (1) and (3) by enforcing that the coproduct $A_1 + \cdots + A_n$ of objects of $\mathscr{D}$ is split into A_1 and $A_2 + \cdots + A_n$.

The following two properties are then sufficient to ensure that applying rules eagerly eventually leads to a coequaliser:

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

7.1 Progress

Progress for the main unification phase means that given a coequalising diagram $\coprod_i Ka_i \overset{t}{\underset{u}{\rightrightarrows}} \Gamma$, there is always one rule that applies, in some sense that can be made precise for each rule, roughly meaning that Γ, t, and u are of the shape Γ', t' and u' (up to isomorphism) when the conclusion of the rule is $\Gamma' \vdash t' = u' \Rightarrow \ldots$, and all the side conditions in the premises are satisfied (as, for example, the only premise of (4)). Similar definitions are required for the pruning phase.

Proposition 2. *Given an input* $\coprod_{i\in I} Ka \overset{t}{\underset{u}{\rightrightarrows}} \Gamma$, *there is one rule of the main unification phase that applies. Given an input*

$$\begin{array}{ccc} \coprod_{i\in I} Ka_i & \xrightarrow{\coprod_i \mathcal{L}Kf_i} & \coprod_{i\in I} Kb_i \\ {\scriptstyle g}\downarrow & & \\ X & & \end{array}$$

there is one rule in the pruning phase that applies.

Remark 9. Essentially, there is exactly one rule that applies, if we ignore the possible reordering of the coproduct in the splitting rules.

7.2 Termination

The usual termination argument for pattern unification applies. Roughly, it consists in defining the size of an input and realising that it strictly decreases in the premises. This relies on the notion of the size $|\Gamma|$ of a context Γ (as an element of $\mathscr{D}^+$), which can be defined without ambiguity as its size as a finite family of elements of $\mathcal{A}$ (see Remark 4). We extend this definition to the case where $\Gamma = \top$, by taking $|\top| = 0$.

We also need to define the size[1] $||t||$ of a *term* $t : Ka \to T\Gamma$ recursively by $||M(f)|| = 1$ and $||o(f; s)|| = 1 + ||f||$, where the size of a list of terms is the sum of the size of each term of the list.

First, we sketch the termination argument for the pruning phase, and then for the main unification phase.

Pruning phase We define the size of a judgement $\Gamma \vdash f :> g \Rightarrow u; \sigma \dashv \Delta$ to be $||f||$. It is straightforward to check that the sizes of the premises are strictly smaller than the size of the conclusion, for each recursive rule of the pruning phase.

[1] This definition can be made formal by providing an adequate F-algebra structure on the constant functor mapping any object of $\mathcal{A}$ to $\mathbb{N}$.

Unification phase We define the size of a judgement $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$ to be the pair $(|\Gamma|, ||t|| + ||u||)$. The following lemma ensures that for each recursive rule of the unification phase, the sizes of the premises are strictly smaller (for the lexicographic order) than the size of the conclusion.

Lemma 12. *For any $t : Ka \to T\Gamma$ and $\sigma : \Gamma \to T\Delta$, if σ is a renaming, i.e, if $\sigma = \mathcal{L}\sigma'$ for some σ', then $||t[\sigma]|| = ||t||$.*

Lemma 13. *If there is a finite derivation tree of $\Gamma \vdash t = u \Rightarrow \sigma \dashv \Delta$, then*

- $|\Gamma| \geq |\Delta|$
- *if $|\Gamma| = |\Delta|$ and $\Delta \neq \top$, then σ is a renaming.*

The proof of the latter lemma relies on the following result.

Lemma 14. *If there is a finite derivation tree of*

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

and $\Delta \neq \top$, then $|\Gamma| = |\Delta|$ and σ is a renaming.

8 Applications

8.1 Linear syntax

Take $\mathcal{A} = \mathbb{N}$ to be the discrete category whose objects are natural numbers. Following Remark (1), a ground context $n \in \mathbb{N}$ specifies the exact number of free variables; a metavariable arity specifies the number of arguments. The linear λ-calculus can be sepecified by the endofunctor defined by $F(X)_n = y1 + \coprod_{p+q=n} X_p \times X_q + (n+1) \times X_{n+1}$ (see [5]).

8.2 Simply-typed second-order syntax

Let T be a set of simple types. Here we consider $\mathcal{A} = \mathbb{F}_m[T] \times T$, where $\mathbb{F}_m[T]$ is the category of finite families indexed by T and injections between them. Let us denote an object (Γ, τ) of $\mathcal{A}$ by $\Gamma \vdash \tau$. Following Remark (1), a metavariable arity $\Gamma \vdash \tau$ specifies the list Γ of input types as well as the output type τ.

If T is the set of simply types for the λ-calculus, generated by a set of base types, then the syntax for simply-typed λ-calculus can be specified by a suitable endofunctor, see e.g. [1].

8.3 Intrinsinc polymorphic syntax

We present intrinsic system F, following [3]. Let $S : \mathbb{F}_m \to \mathrm{Set}$ mapping n to the set Sn of types for system F taking free type variables in $\{0, \ldots, n-1\}$. Now, consider the category $\mathcal{A}$ of contexts of the shape $n; \sigma_1, \ldots, \sigma_p \vdash \tau$, where $\sigma_i, \tau \in Sn$. A morphism between $n, \Gamma \vdash \tau$ and $n', \Gamma' \vdash \tau'$ is a monomorphism $\sigma : n \to n'$ and renamings $\Gamma[\sigma] \to \Gamma'$ such that $\tau[\sigma] = \tau'$. More formally, $\mathcal{A}$ is the oplax colimit of $n \mapsto \mathbb{F}_m[Sn] \times Sn$. The intrinsic syntax of system F can then be defined by a suitable endofunctor, see [3]. Following Remark (1), a metavariable arity $n; \sigma_1, \ldots, \sigma_p \vdash \tau$ specifies the number of type arguments, the list of input types, and the output type.

References

1. M. P. Fiore and C.-K. Hur. Second-order equational logic. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010)*, 2010.
2. Joseph A. Goguen. What is unification? - a categorical view of substitution, equation and solution. In *Resolution of Equations in Algebraic Structures, Volume 1: Algebraic Techniques*, pages 217–261. Academic, 1989.
3. Makoto Hamana. Polymorphic abstract syntax via grothendieck construction. 2011.
4. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.
5. Miki Tanaka. Abstract syntax and variable binding for linear binders. In *MFCS '00*, volume 1893 of *LNCS*. Springer, 2000.
6. Andrea Vezzosi and Andreas Abel. A categorical perspective on pattern unification. *RISC-Linz*, page 69, 2014.