Higher-Order Dynamic Pattern Unification for Dependent Types and Records

Andreas Abel[1] and Brigitte Pientka[2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität, München, Deutschland
andreas.abel@ifi.lmu.de
[2] School of Computer Science, McGill University, Montreal, Canada
bpientka@cs.mcgill.ca

Abstract. While higher-order pattern unification for λ^{Π}-calculus is decidable and unique unifiers exists, we face several challenges in practice: 1) the pattern fragment itself is too restrictive for many applications; this is typically addressed by solving sub-problems which satisfy the pattern restriction eagerly but delay solving sub-problems which are non-patterns until we have accumulated more information. This leads to a dynamic pattern unification. 2) Many systems implement $\lambda^{\Pi\Sigma}$ calculus and hence the known pattern unification algorithms for λ^{Π} are too restrictive.

In this paper, we present a constraint-based unification algorithm for $\lambda^{\Pi\Sigma}$-calculus which solves a richer class of patterns than currently possible; in particular it takes into account type isomorphisms to translate unification problems containing Σ-types into problems only involving Π-types. We prove correctness of our algorithm and discuss its application.

1 Introduction

Higher-order unification is a key operation in logical frameworks, dependently-typed programming systems, or proof assistants supporting higher-order logic. It plays a central role in type inference and reconstruction algorithms, in the execution of programs in higher-order logic programming languages, and in reasoning about the totality of functions defined by pattern-matching clauses.

While full higher-order unification is undecidable [7], Miller [8] identified a decidable fragment of higher-order unification problems, called the *pattern* fragment. A pattern is a unification problem where all meta-variables (or logic variables) occurring in a term are applied to some distinct bound variables. For example, the problem $\lambda x\, y\, z.\, X\, x\, y = \lambda x\, y\, z.\, x\, (\mathsf{suc}\, y)$ falls into the pattern fragment, because the meta-variable X is applied to distinct bound variables x and y; the pattern condition allows us to solve the problem by a simple abstraction $X = \lambda x\, y.\, x\, (\mathsf{suc}\, y)$. This is not possible for non-patterns; examples for non-pattern problems which have no unique most general unifier can be obtain by changing the left hand side of the previous problem to $\lambda x\, y\, z.\, X\, x\, x\, y$ (non-linearity), $\lambda x\, y\, z.\, X\, (Y\, x)\, y$ (X applied to another meta-variable) or $\lambda x\, y\, z.\, X\, x\, (\mathsf{suc}\, y)$ (X applied to non-variable term).

In practice we face several challenges: First, the pattern fragment is too restrictive for many applications and we want to solve more general problems incrementally. Systems such as Twelf [12], Beluga [14], and Delphin [15] solve eagerly sub-problems which fall into the pattern fragment and delay sub-problems outside the pattern fragment until more information has been gathered which in turn simplifies the delayed sub-problems. The meta-theory justifying the correctness of such a strategy is largely unexplored and complex (an exception is the work by Reed [16]).

Second, we often want to consider richer calculi beyond the λ^{Π}-calculus. In Beluga and Twelf for example we use Σ-types to group assumptions together. In Agda [10], we support Σ-types in form of records with associated η-equality in its general form. Yet, little work has been done on extending the pattern fragment to handle also Σ-types. The following terms may be seen as equivalent: (a) $\lambda y_1.\lambda y_2.\, X\,(y_1, y_2)$, (b) $\lambda y.\, X\,(\mathsf{fst}\ y)\ (\mathsf{snd}\ y)$ and (c) $\lambda y_1.\lambda y_2.\, X\, y_1\, y_2$. Only the last term falls within the pattern fragment as originally described by Miller. However, the other two terms can be transformed such that they also fall into the pattern fragment: for term (a), we replace X with $\lambda y_1.\lambda y_2.\, X'\, y_1\, y_2$; for term (b), we unfold y which stands for a pair and replace y with (y_1, y_2).

In this paper, we describe a higher-order unification algorithm for the $\lambda^{\Pi\Sigma}$ calculus; our algorithm handles lazily η-expansion and we translate terms into the pure pattern fragment where a meta-variable is applied to distinct bound variables. The key insight is to take into account type isomorphisms for Σ, the dependently typed pairs: $\Pi z{:}(\Sigma x{:}A.B).C$ is isomorphic to $\Pi x{:}A.\Pi y{:}B.[(x, y)/z]C$, and a function $f{:}\Pi x{:}A.\Sigma y{:}B.C$ can be translated into two functions $f_1 : \Pi x{:}A.B$ and $f_2 : \Pi x{:}A.[f_1\, x/y]C$. These transformations allow us to handle a richer class of dependently-typed patterns than previously considered.

Following Nanevski et al. [9] and Pientka [13], our description takes advantage of modelling meta-variables as closures; instead of directly considering a meta-variable X at function type $\Pi \boldsymbol{x{:}A}.B$ which is applied to $\boldsymbol{x}$, we describe them as contextual objects, i.e., objects of type B in a context $\boldsymbol{x{:}A}$, which are associated with a delayed substitution for the local context $\boldsymbol{x{:}A}$.[3] This allows us to give a high-level description and analysis following Dowek et al. [2], but not resorting to explicit substitutions; more importantly, it provides a logical grounding for some of the techniques such as "pre-cooking" and handles a richer calculus including Σ-types. Our constraint-based unification algorithm also avoids some of the other shortcomings; as pointed out by Reed [16], the algorithm sketched in Dowek et al. [2] fails to terminate on some inputs. We give a clear specification of the pruning operation which eliminates bound variable dependencies for the dependently typed case and show correctness of the unification algorithms following Reed [16] in three steps: 1) we show it terminates, 2) we show that the transformations in our unification algorithm preserve types, and 3) that each transition neither destroys nor creates (additional) solutions.

Our work is to our knowledge the first comprehensive description of constraint-based higher-order pattern unification for the $\lambda^{\Pi\Sigma}$ calculus. It builds on and

[3] We write $\boldsymbol{x{:}A}$ for a vector $x_1{:}A_1, \ldots x_n{:}A_n$.

extends prior work by Reed [16] to handle Σ-types. Previously, Elliot [4] described unification for Σ-types in a Huet-style unification algorithm. While it is typically straightforward to incorporate η-expansions and lowering for meta-variables of Σ-type [18, 11], there is little work on extending the notion of Miller patterns to be able to handle meta-variables which are applied to projections of bound variables. Fettig and Löchner [5] describe a higher-order pattern unification algorithm with finite products in the simply typed lambda-calculus. Their approach does not directly exploit isomorphisms on types, but some of the ideas have a similar goal: for example abstractions $\lambda x.\,\mathsf{fst}\,x$ is translated into $\lambda(x_1, x_2).\,\mathsf{fst}\,(x_1\,,\,x_2)$ which in turn normalizes to $\lambda(x_1, x_2).x_1$ to eliminate projections. Duggan [3] also explores extended higher-order patterns for products in the simply-typed setting; he generalizes Miller's pattern restriction for the simply-typed lambda-calculus by allowing repeated occurrences of variables to appear as arguments to meta-variables, provided such variables are prefixed by distinct sequences of projections.

Our work has been already tested in practice. Some of the ideas described in this paper are incorporated into the implementation of the dependently-typed Beluga language; in Beluga, Σ-types occur in a restricted form, i.e., only variable declarations in contexts can be of Σ-type and there is no nesting of Σ-types.

Due to space restrictions, most proofs have been omitted; they can be found in an extended version of this article on the authors' homepages.

2 $\lambda^{\Pi\Sigma}$-calculus

In this paper, we are considering an extension of the $\lambda^{\Pi\Sigma}$-calculus with meta-variables. Meta-variables are characterized as a closure $u[\sigma]$ which is the use of the meta-variable u under the suspended explicit substitution σ. The previous term $\lambda x\,y\,z.\,X\,x\,y$ with the meta-variable X which has type $\Pi x{:}A.\Pi y{:}B.C$ is represented in our calculus as $\lambda x\,y\,z.\,u[x, y]$ where u has type $C[x{:}A, y{:}B]$ and $[x, y]$ is a substitution with domain $x{:}A, y{:}B$ and the range x, y, z. Instead of an abstraction, we can directly replace u with a closed object $x, y.\,x\,(\mathsf{suc}\,y)$. This eliminates the need to craft a λ-prefix for the instantiation of meta-variables, avoids spurious reductions, and provides simple justifications for techniques such as lowering. In general, the meta-variable u stands for a contextual object $\hat{\Psi}.M$ where $\hat{\Psi}$ describes the ordinary bound variables which may occur in M. This allows us to rename the free variables occurring in M if necessary. We use the following convention: If the meta-variable u is associated with the identity substitution, we simply write u instead of $u[\mathsf{id}]$. A meta-variable u has the contextual type $A[\Psi]$ thereby characterizing an object of type A in the context Ψ. Our grammar and our subsequent typing rules enforce that objects are β-normal; this will simplify the later development.

The grammar is mostly straightforward; a signature Σ is a collection of constant declarations, which take one of the forms $\mathbf{a} : \kappa$ (type family declaration) or $\mathbf{c} : A$ (constructor declaration). Because variable substitutions ρ play a special role in the formulation of our unification algorithm, we recognize them as

Variables	x, y, z
Meta variables	u, v, w
Sorts	$s ::= \mathsf{type} \mid \mathsf{kind}$
Atomic types	$P, Q ::= \mathbf{a}\,\boldsymbol{M}$
Types	$A, B, C, D ::= P \mid \Pi x{:}A.B \mid \Sigma x{:}A.B$
Kinds	$\kappa ::= \mathsf{type} \mid \Pi x{:}A.\kappa$
(Rigid) heads	$H ::= \mathbf{a} \mid \mathbf{c} \mid x$
Projections	$\pi ::= \mathsf{fst} \mid \mathsf{snd}$
Evaluation contexts	$E ::= \bullet \mid E\,N \mid \pi\,E$
Neutral terms	$R ::= E[H] \mid E[u[\sigma]]$
Normal terms	$M, N ::= R \mid \lambda x.M \mid (M\ ,\ N)$
Substitutions	$\sigma, \tau ::= \cdot \mid \sigma, M$
Variable substitutions	$\rho, \xi ::= \cdot \mid \rho, x$
Contexts	$\Psi, \Phi, \Gamma ::= \cdot \mid \Psi, x{:}A$
Signature	$\Sigma ::= \cdot \mid \Sigma, \mathbf{a}{:}\kappa \mid \Sigma, \mathbf{c}{:}A$
Meta substitutions	$\theta, \eta ::= \cdot \mid \theta, \hat{\Psi}.M/u$
Meta contexts	$\Delta ::= \cdot \mid \Delta, u{:}A[\Psi]$

Fig. 1. $\lambda^{\Pi\Sigma}$ with meta-variables

a subclass of general substitutions σ. Identity substitutions id_Φ are defined recursively by $\mathsf{id}_{\cdot} = (\cdot)$ and $\mathsf{id}_{\Phi,x:A} = (\mathsf{id}_\Phi, x)$. The subscript Φ is dropped when unambiguous. If Φ is a sub-context of Ψ (in particular if $\Psi = \Phi$) then id_Φ is a well-formed substitution in Ψ, i.e., $\Psi \vdash \mathsf{id}_\Phi : \Phi$ holds (see Fig. 2). We write $\hat{\Phi}$ for the list of variables $\mathsf{dom}(\Phi)$ in order of declaration.

We write $E[M]$ for plugging term M into the hole $\bullet$ of evaluation context E. This will be useful when describing the unification algorithm, since we often need to have access to the head of a neutral term. In the λ^Π-calculus, this is often achieved using the spine notation [1] simply writing $H\,M_1 \ldots M_n$. Evaluation contexts are the proper generalization of spines to projections.

Occurrences and free variables. If α, β are a syntactic entities such as evaluation context, term, or substitution, $\alpha, \beta ::= E \mid R \mid M \mid \sigma$, we write $\alpha\{\beta\}$ if β is a part of α. If we subsequently write $\alpha\{\beta'\}$ then we mean the replacement of the indicated occurrence of β by β'. We say an occurrence is *rigid* if it is not part of a delayed substitution σ of a meta-variable, otherwise it is termed *flexible*. For instance, in $\mathbf{c}\,(u[y_1])\,(x_1\,x_2)\,(\lambda z.\,z\,x_3\,v[y_2, w[y_3]])$ there are rigid occurrences of $x_{1..3}$ and flexible occurrences of $y_{1..3}$. The meta-variables u, v appear in a rigid and w in a flexible position. A rigid occurrence is *strong* if it is not in the evaluation context of a free variable. In our example, only x_2 does *not* occur strongly rigidly. Following Reed [16] we indicate rigid occurrences by $\alpha\{\beta\}^{\mathsf{rig}}$ and strongly rigid occurrences by $\alpha\{\beta\}^{\mathsf{srig}}$.

We denote the set of free variables of α by $\mathsf{FV}(\alpha)$ and the set of free meta variables by $\mathsf{FMV}(\alpha)$. A superscript $^{\mathsf{rig}}$ indicates to count only the rigidly occurring variables.

Neutral Terms/Types $\boxed{\Delta;\Psi \vdash R \Rightarrow A}$

$$\frac{\Sigma(\mathbf{a}) = \kappa}{\Delta;\Psi \vdash \mathbf{a} \Rightarrow \kappa} \quad \frac{\Sigma(\mathbf{c}) = A}{\Delta;\Psi \vdash \mathbf{c} \Rightarrow A} \quad \frac{\Psi(x) = A}{\Delta;\Psi \vdash x \Rightarrow A} \quad \frac{u{:}A[\Phi] \in \Delta \quad \Delta;\Psi \vdash \sigma \Leftarrow \Phi}{\Delta;\Psi \vdash u[\sigma] \Rightarrow [\sigma]_\Phi A}$$

$$\frac{\Delta;\Psi \vdash R \Rightarrow \Pi x{:}A.B \quad \Delta;\Psi \vdash M \Leftarrow A}{\Delta;\Psi \vdash R\,M \Rightarrow [M/x]_A B}$$

$$\frac{\Delta;\Psi \vdash R \Rightarrow \Sigma x{:}A.B}{\Delta;\Psi \vdash \mathsf{fst}\,R \Rightarrow A} \quad \frac{\Delta;\Psi \vdash R \Rightarrow \Sigma x{:}A.B}{\Delta;\Psi \vdash \mathsf{snd}\,R \Rightarrow [\mathsf{fst}\,R/x]_A B}$$

Normal Terms $\boxed{\Delta;\Psi \vdash M \Leftarrow A}$

$$\frac{\Delta;\Psi \vdash R \Rightarrow A \quad A =_\eta C}{\Delta;\Psi \vdash R \Leftarrow C}$$

$$\frac{\Delta;\Psi, x{:}A \vdash M \Leftarrow B}{\Delta;\Psi \vdash \lambda x.M \Leftarrow \Pi x{:}A.B} \quad \frac{\Delta;\Psi \vdash M \Leftarrow A \quad \Delta;\Psi \vdash N \Leftarrow [M/x]_A B}{\Delta;\Psi \vdash (M\,,\,N) \Leftarrow \Sigma x{:}A.B}$$

Substitutions $\boxed{\Delta;\Psi \vdash \sigma \Leftarrow \Psi'}$

$$\frac{}{\Delta;\Psi \vdash \cdot \Leftarrow \cdot} \quad \frac{\Delta;\Psi \vdash \sigma \Leftarrow \Psi' \quad \Delta;\Psi \vdash M \Leftarrow [\sigma]_{\Psi'} A}{\Delta;\Psi \vdash \sigma, M \Leftarrow \Psi', x{:}A}$$

LF Types and Kinds $\boxed{\Delta;\Psi \vdash A \Leftarrow s}$

$$\frac{\Delta;\Psi \vdash P \Rightarrow \mathsf{type}}{\Delta;\Psi \vdash P \Leftarrow \mathsf{type}} \quad \frac{\Delta;\Psi \vdash A \Leftarrow \mathsf{type} \quad \Delta;\Psi, x{:}A \vdash B \Leftarrow \mathsf{type}}{\Delta;\Psi \vdash \Sigma x{:}A.B \Leftarrow \mathsf{type}}$$

$$\frac{}{\Delta;\Psi \vdash \mathsf{type} \Leftarrow \mathsf{kind}} \quad \frac{\Delta;\Psi \vdash A \Leftarrow \mathsf{type} \quad \Delta;\Psi, x{:}A \vdash B \Leftarrow s}{\Delta;\Psi \vdash \Pi x{:}A.B \Leftarrow s}$$

Meta-Substitutions $\boxed{\Delta \vdash \theta \Leftarrow \Delta'}$

$$\frac{\text{for all } u{:}A[\Phi] \in \Delta' \text{ and } \hat{\Phi}.M/u \in \theta \quad \Delta; [\![\theta]\!]\Phi \vdash M \Leftarrow [\![\theta]\!]A}{\Delta \vdash \theta \Leftarrow \Delta'}$$

Meta-Context $\boxed{\vdash \Delta\ \mathsf{mctx}}$

$$\frac{\text{for all } u{:}A[\Psi] \in \Delta \quad \Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type}}{\vdash \Delta\ \mathsf{mctx}}$$

Fig. 2. Typing rules for LF with meta-variables

Typing rules are given in Figure 2. We have record types $\Sigma x{:}A.\,B$ but no record kinds $\Sigma x{:}A.\,\kappa$. Our typing rules ensure that terms are in β-normal form, but they need not be η-long. The judgment $A =_\eta C$ (rules omitted) compares A and C modulo η, i.e., modulo $R = \lambda x.\,R\,x$ ($x \notin \mathsf{FV}(R)$) and $R = (\mathsf{fst}\,R\,,\,\mathsf{snd}\,R)$.

Hereditary substitution. For α a well-typed entity in context Ψ and $\Delta;\Phi \vdash \sigma : \Psi$ a well-formed substitution, we facilitate a simultaneous substitution operation $[\sigma]_\Psi(\alpha)$ that substitutes the terms in σ for the variables as listed by Ψ in α and produces a β-normal result. Such an operation exists for well-typed terms, since $\lambda^{\Pi\Sigma}$ is normalizing. A naive implementation just substitutes and then normalizes, a refined implementation, called *hereditary substitution* [19], proceeds by resolving created redexes on the fly through new substitutions. Details can be found in Nanevski et al. [9], but do not concern us much here. Single substitution $[N/x]_A(\alpha)$ is conceived as a special case of simultaneous substitution. The type annotation A and the typing information in Ψ allow hereditary substitution to be defined by structural recursion; if no ambiguity arises, we may omit indices Ψ and A from substitutions.

Meta-substitution. The two classes of variables, ordinary variables declared in the context Ψ and meta-variables declared in the meta-context Δ, give rise to two different substitution operations: the hereditary substitution defined previously and the meta-substitution defined in this section. The single meta-substitution operation is written as $[\![\hat{\Psi}.M/u]\!]_{A[\Psi]}(N)$ and the simultaneous meta-substitution is written as $[\![\theta]\!]_{\Delta}(N)$. Subsequently, we define the application of the single meta-substitution to a given term and type, but the simultaneous meta-substitution definition can be easily derived from it. When we apply $\hat{\Psi}.M/u$ to $u[\sigma]$ we first substitute $\hat{\Psi}.M$ for u in the substitution σ to obtain σ'. Subsequently, we continue to apply σ' to M hereditarily to obtain M'. As we annotate meta-substitutions with their type, we can appropriately annotate σ' with its domain Ψ to obtain M'. Without annotating meta-substitution with the type $C[\Psi]$, we would not be able annotate the operation $[\sigma']M$ appropriately. We omit the typing annotation in the subsequent development for better readability. Because M' may not be neutral, we may trigger a β-reduction and we return M' together with its (approximate) type C.

The typing rules ensure that the type of the instantiation $\hat{\Psi}.M$ and the type of u agree, i.e. we can replace u which has type $A[\Psi]$ with a normal term M if M has type A in the context Ψ. Because of α-conversion, the variables that are substituted at different occurrences of u may be different, and we write $\hat{\Psi}.M$ where $\hat{\Psi}$ binds all the free variables in M. We can always appropriately rename the bound variable in $\hat{\Psi}$ such that they match the domain of the postponed substitution σ'. This complication can be eliminated in an implementation of the calculus based on de Bruijn indexes. Applying the meta-substitution to an LF object will terminate for the same reasons as the ordinary substitution operation terminates; either we apply the substitution to a sub-expression or the objects we substitute are smaller. For an in depth discussion, we refer the reader to Nanevski et al. [9].

3 Constraint-based unification

We define the unification algorithm using rewrite rules which solve constraints incrementally. Constraints are defined as follows:

Meta-substitution on normal terms / types

$$
\begin{array}{lll}
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\Pi x{:}A.B) & = \Pi x{:}A'.B' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(A) = A' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(B) = B' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\Sigma x{:}A.B) & = \Sigma x{:}A'.B' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(A) = A' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi}(B) = B' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\mathsf{type}) & = \mathsf{type} & \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\lambda x.N) & = \lambda x.N' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N) = N' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N_1, N_2) & = (N_1', N_2') & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N_2) = N_2' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N_2) = N_2' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) & = N & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}R = N : A \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) & = R' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}R = R' \\
\llbracket\hat{\Psi}.M/x\rrbracket_{C[\Psi]}(N) & \quad \text{fails} & \text{otherwise}
\end{array}
$$

Meta-substitution on neutral terms

$$
\begin{array}{lll}
\llbracket\hat{\Psi}.M/u\rrbracket_{A[\Psi]}(u[\sigma]) & = M' : A & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\sigma) = \sigma' \\
 & & \text{and } [\sigma']_\Psi(M) = M' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{A[\Psi]}(v[\sigma]) & = v[\sigma'] & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\sigma) = \sigma' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R\,N) & = R'\,N' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) = R' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N) = N' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R\,N) & = M'' : B & \text{if } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) = \lambda y.M' : \Pi x{:}A.B \\
 & & \text{where } \Pi x{:}A.B \leq A,\ N' = \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(N) \\
 & & \text{and } M'' = [N'/y]_A(M') \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\pi R) & = \pi R' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) = R' \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\mathsf{fst}\,R) & = N_1 : A & \text{if } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) = (N_1, N_2) : \Sigma x{:}A.B \\
 & & \text{where } \Sigma x{:}A.B \leq C \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\mathsf{snd}\,R) & = N_2 : B & \text{if } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(R) = (N_1, N_2) : \Sigma x{:}A.B \\
 & & \text{where } \Sigma x{:}A.B \leq C \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(x) & = x & \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\mathbf{c}) & = \mathbf{c} & \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\mathbf{a}) & = \mathbf{a} & \\
\llbracket\hat{\Psi}.M/x\rrbracket_{C[\Psi]}(R) & \quad \text{fails} & \text{otherwise}
\end{array}
$$

Meta-substitution on substitutions

$$
\begin{array}{lll}
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\cdot) & = \cdot & \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\sigma, M) & = \sigma', M' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\sigma) = \sigma' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(M) = M' \\
\llbracket\hat{\Psi}.M/x\rrbracket_{C[\Psi]}(\sigma) & \quad \text{fails} & \text{otherwise}
\end{array}
$$

Meta-substitution on contexts

$$
\begin{array}{lll}
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\cdot) & = \cdot & \\
\llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\Psi, x{:}A) & = \Psi', x{:}A' & \text{where } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(\Psi) = \Psi' \\
 & & \text{and } \llbracket\hat{\Psi}.M/u\rrbracket_{C[\Psi]}(A) = A' \\
\llbracket\hat{\Psi}.M/x\rrbracket_{C[\Psi]}(\Psi) & \quad \text{fails} & \text{otherwise}
\end{array}
$$

Fig. 3. Meta-substitution

Constraint	$K ::= \top \mid \bot$	trivial constraint and inconsistency
	$\mid\ \Psi \vdash M = N : C$	unify term M with N
	$\mid\ \Psi \mid R{:}A \vdash E = E'$	unify evaluation context E with E'
	$\mid\ \Psi \vdash u \leftarrow M : C$	solution for u found
C. sets	$\mathcal{K} ::= K \mid \mathcal{K} \wedge K$	modulo laws of conjunction.

Our basic constraints are of the form $\Psi \vdash M = N : C$. The type annotation $\Psi \vdash C$ serves two purposes: First, it is necessary to ensure that all substitutions created and used in our transformations can be properly annotated and hence we can use the fact that their application will terminate and produce again normal forms. Second, the type annotations in the context Ψ are necessary to eliminate Σ-types. For both purposes, simple types, i.e., the dependency-erasure of $\Psi \vdash C$ would suffice. However, we keep dependency in this presentation to scale this work from $\lambda^{\Pi\Sigma}$ to non-erasable dependent types such as in Agda.

A unification problem is described by $\Delta \Vdash \mathcal{K}$ where Δ contains the typings of all the meta variables in $\mathcal{K}$. A meta-variable u is *solved*, if there is a constraint $\Psi \vdash u \leftarrow M : C$ in $\mathcal{K}$; otherwise we call u *active*. A solved metavariable does not appear in any other constraints nor in any type in Δ (nor in its solution M).

Intuitively, a set of constraints is well-formed if each constraint $\Psi \vdash M = N : C$ is well typed. Unfortunately, this is complicated by the fact that we may delay working on some sub-terms; to put it differently, we can work on subterms in an arbitrary order. Yet, the type of an equation may depend on the solvability of another postponed equation. Consider for example tuples. If (M_1, M_2) and (N_1, N_2) both have type $\Sigma x{:}A.B$, then M_1 and N_1 have type A. However, types may get out of sync when we consider M_2 and N_2. M_2 has type $[M_1/x]B$ while N_2 has type $[N_1/x]B$, and we only know that their types agree, if we know that M_1 is equal to N_1. Similar issues arise for function types and applications. Following Reed [16] we adopt here a weaker typing invariant, namely typing modulo constraints.

3.1 Typing modulo

For all typing judgments $\Delta; \Psi \vdash J$ defined previously, we define $\Delta; \Psi \vdash_{\mathcal{K}} J$ by the same rules as for $\Delta; \Psi \vdash J$ except replacing syntactic equality $=$ with $=_{\mathcal{K}}$. We write $\hat{\Psi}.M =_{\mathcal{K}} N$ if for any ground meta-substitution θ that is a ground solution for $\mathcal{K}$, we have $[\![\theta]\!]M = [\![\theta]\!]N$. To put it differently, if we can solve $\mathcal{K}$, we can establish that M is equal to N.

The following lemmas proven by Reed [16] hold also for the extension to Σ-types; we keep in mind that the judgment J stands for either a typing judgment or an equality judgment.

Lemma 1 (Substitution principle modulo). *If $\Delta; \Psi \vdash_{\mathcal{K}} M \Leftarrow A$ and $\Psi, x{:}B, \Psi' \vdash_{\mathcal{K}} J$ and $A =_{\mathcal{K}} B$ then $\Delta; \Psi, [M/x]_A(\Psi') \vdash_{\mathcal{K}} [M/x]_A(J)$.*

Lemma 2 (Meta-substitution principle modulo).

– *If* $\Delta;\Psi \vdash_{\mathcal{K}} M \Leftarrow A$ *and* $u{:}B[\Psi'] \in \Delta$ *and* $A[\Psi] =_{\mathcal{K}} B[\Psi']$ *and* $\Delta;\Phi \vdash_{\mathcal{K}} J$ *then* $[\![\hat{\Psi}.M/u]\!]\Delta; [\![\hat{\Psi}.M/u]\!]\Psi \vdash_{[\![\hat{\Psi}.M/u]\!]\mathcal{K}} [\![\hat{\Psi}.M/u]\!]J$.
– *If* $\Delta_1 \vdash_{\mathcal{K}} \theta \Leftarrow \Delta_0$ *and* $\Delta_0;\Phi \vdash_{\mathcal{K}} J$ *then* $\mathcal{K}' = [\![\theta]\!]\mathcal{K}$ *and* $\Delta_1; [\![\theta]\!]\Phi \vdash_{\mathcal{K}'} [\![\theta]\!]J$.

Proof. Induction on the derivation $\Delta;\Phi \vdash_{\mathcal{K}} J$. All cases are by inversion, appeal to i.h., and reassembling the result. We show the case where we transition between checking and synthesizing a type.

$$\textit{Case}\ \ \mathcal{D} = \frac{\Delta;\Phi \vdash_{\mathcal{K}} R \Rightarrow C_1 \quad C_1 =_{\mathcal{K}} C_2}{\Delta;\Phi \vdash_{\mathcal{K}} R \Leftarrow C_2}$$

let $\theta = \hat{\Psi}.M/u$. We omit the type annotation $A[\Psi]$ when we apply θ.
$[\![\theta]\!]\Delta; [\![\theta]\!]\Phi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]R \Rightarrow [\![\theta]\!]C_1$ by i.h.
$[\![\theta]\!]C_1 =_{\mathcal{K}'} [\![\theta]\!]C_2$ by i.h. $C_1 =_{\mathcal{K}} C_2$
$[\![\theta]\!]\Delta; [\![\theta]\!]\Phi' \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]R \Leftarrow [\![\theta]\!]C_2$. □

Intuitively, a unification problem $\Delta \Vdash \mathcal{K}$ is well-formed if all constraints $(\Psi \vdash M = N : C) \in \mathcal{K}$ are well-typed modulo $\mathcal{K}$, i.e., $\Delta;\Psi \vdash_{\mathcal{K}} M \Leftarrow C$ and $\Delta;\Psi \vdash_{\mathcal{K}} N \Leftarrow C$. We will come back to this later when we prove correctness of our algorithm, but it is helpful to keep the typing invariant in mind when explaining the transitions in our algorithm.

3.2 A higher-order dynamic pattern unification algorithm for dependent types and records

The higher-order dynamic pattern unification algorithm is presented as rewrite rules on the set of constraints $\mathcal{K}$ in meta variable context Δ. The *local simplification rules* (Figure 4) apply to a single constraint, decomposing it and molding it towards a pattern by η-contraction and projection elimination. Decomposition of a neutral term is defined using evaluation contexts to have direct access to the head.

The other *unification steps* (Figure 5) work on a meta-variable and try to find an instantiation for it. Lowering rules transform a meta-variable of higher type to one of lower type. Flattening Σ-types concentrates on a meta-variable $u{:}A[\Phi]$ and eliminates Σ-types from the context Φ. The combination of the flattening Σ-types transition and the eliminating projections transition allow us to transform a unification problem into one which resembles our traditional pattern unification problem. The pruning transition is explained in detail in Section 3.4 and unifying a meta-variable with itself is discussed in Section 3.5.

Our algorithm can deal with a larger class of patterns where we require that meta-variables are associated with a linear substitution. To motivate our rules, let us consider some problems $\Psi \vdash u[\sigma] = M : C$ that fall out of the Miller pattern fragment, meaning that σ is not a list of disjoint variables. We may omit types and/or context if appropriate.

η-contraction $u[\lambda x.\, y\, (\mathsf{fst}\, x\, ,\ \mathsf{snd}\, x)] = M$
Solved by contracting the l.h.s. to $u[y]$.

Decomposition of functions		
$\Psi \vdash \lambda x.M = \lambda x.N : \Pi x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi, x{:}A \vdash M = N : B$
$\Psi \vdash \lambda x.M = R : \Pi x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi, x{:}A \vdash M = R\,x : B$
$\Psi \vdash R = \lambda x.M : \Pi x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi, x{:}A \vdash R\,x = M : B$
Decomposition of pairs		
$\Psi \vdash (M_1\,, M_2) = (N_1\,, N_2) : \Sigma x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi \vdash M_1 = N_1 : A \wedge \Psi \vdash M_2 = N_2 : [M_1/x]B$
$\Psi \vdash (M_1\,, M_2) = R : \Sigma x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi \vdash M_1 = \mathsf{fst}\, R : A \wedge \Psi \vdash M_2 = \mathsf{snd}\, R : [M_1/x]B$
$\Psi \vdash R = (M_1\,, M_2) : \Sigma x{:}A.\, B$	$\mapsto_{\mathsf{d}}$	$\Psi \vdash \mathsf{fst}\, R = M_1 : A \wedge \Psi \vdash \mathsf{snd}\, R = M_2 : [\mathsf{fst}\, R/x]B$
Decomposition of neutrals		
$\Psi \vdash E[H] = E'[H] : C$	$\mapsto_{\mathsf{d}}$	$\Psi \mid H : A \vdash E = E'$ where $\Psi \vdash H \Rightarrow A$
$\Psi \vdash E[H] = E'[H'] : C$	$\mapsto_{\mathsf{d}}$	$\bot$ if $H \neq H'$
Decomposition of evaluation contexts		
$\Psi \mid R : A \vdash \bullet = \bullet$	$\mapsto_{\mathsf{d}}$	$\top$
$\Psi \mid R : \Pi x{:}A.\, B \vdash E[\bullet\, M] = E'[\bullet\, M']$	$\mapsto_{\mathsf{d}}$	$\Psi \vdash M = M' : A \wedge \Psi \mid R\,M : [M/x]B \vdash E = E'$
$\Psi \mid R : \Sigma x{:}A.\, B \vdash E[\mathsf{fst}\,\bullet] = E'[\mathsf{fst}\,\bullet]$	$\mapsto_{\mathsf{d}}$	$\Psi \mid \mathsf{fst}\, R\ : A \vdash E = E'$
$\Psi \mid R : \Sigma x{:}A.\, B \vdash E[\mathsf{snd}\,\bullet] = E'[\mathsf{snd}\,\bullet]$	$\mapsto_{\mathsf{d}}$	$\Psi \mid \mathsf{snd}\, R : [\mathsf{fst}\, R/x]B \vdash E = E'$
$\Psi \mid R : \Sigma x{:}A.\, B \vdash E[\pi\,\bullet] = E'[\pi'\,\bullet]$	$\mapsto_{\mathsf{d}}$	$\bot$ if $\pi \neq \pi'$
Orientation		
$\Psi \vdash M = u[\sigma] : C$ with $M \neq v[\tau]$	$\mapsto_{\mathsf{d}}$	$\Psi \vdash u[\sigma] = M : C$
η-Contraction		
$\Psi \vdash u[\sigma\{\lambda x.\, R\,x\}] = N : C$	$\mapsto_{\mathsf{e}}$	$\Psi \vdash u[\sigma\{R\}] = N : C$
$\Psi \vdash u[\sigma\{(\mathsf{fst}\, R, \mathsf{snd}\, R)\}] = N : C$	$\mapsto_{\mathsf{e}}$	$\Psi \vdash u[\sigma\{R\}] = N : C$
Eliminating projections		
$\Psi_1, x : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, \Sigma z{:}B.C,\ \Psi_2$ $\vdash u[\sigma\{\pi\,(x\,\boldsymbol{M})\}] = N : D$ where $\pi \in \{\mathsf{fst}, \mathsf{snd}\}$	$\mapsto_{\mathsf{p}}$	$\Psi_1, x_1 : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, B,\ x_2 : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, [(x_1\,\boldsymbol{y})/z]C,\ \Psi_2$ $\vdash u[[\tau]\sigma] = [\tau]N : [\tau]D$ where $\tau = [\lambda \boldsymbol{y}.\, (x_1\,\boldsymbol{y},\, x_2\,\boldsymbol{y})/x]$

Fig. 4. Local simplification.

Eliminating projections $y : \Pi x{:}A.\, \Sigma z{:}B.\, C \vdash u[\lambda x.\, \mathsf{fst}\,(y\,x)] = M$
Applying substitution $\tau = [\lambda x.\, (y_1\,x\,,\, y_2\,x)/y]$ yields problem $y_1 : \Pi x{:}A.\, B$, $y_2 : \Pi x{:}A.\, [y_1\,x/z]C \vdash u[\lambda x.\, y_1\,x] = [\tau]M$ which is solved by η-contraction, provided $y_2 \notin \mathsf{FV}([\tau]M)$.

Lowering $u : (\Sigma x{:}A.\, B)[\Phi] \Vdash \mathsf{fst}\,(u[y]) = \mathsf{fst}\, y$
This equation determines only the first component of the tuple u. Thus, decomposition into $u[y] = y$, which also determines the second component, loses solutions. Instead we replace u by a pair (u_1, u_2) of meta-variables of *lower* type, yielding $u_1 : A[\Phi], u_2 : ([u_1/x]B)[\Phi] \Vdash u_1[y] = \mathsf{fst}\, y$.

Local simplification

$\Delta \Vdash \mathcal{K} \wedge K \mapsto \Delta \Vdash \mathcal{K} \wedge \mathcal{K}'$ if $K \mapsto_m \mathcal{K}'$ $(m \in \{\mathsf{d}, \mathsf{e}, \mathsf{p}\})$

Instantiation (notation)

$\Delta \Vdash \mathcal{K} + (\Phi \vdash u \leftarrow M : A)$ $= [\![\theta]\!]\Delta \Vdash [\![\theta]\!]\mathcal{K} \;\wedge\; [\![\theta]\!]\Phi \vdash u \leftarrow M : [\![\theta]\!]A$
where $\theta = \hat{\Phi}.M/u$

Lowering

$\Delta \Vdash \mathcal{K}$ $\mapsto \Delta, v{:}B[\Phi, x{:}A] \Vdash \mathcal{K}$
$u{:}(\Pi x{:}A.B)[\Phi] \in \Delta$ active $+ \Phi \vdash u \leftarrow \lambda x.v : \Pi x{:}A.\, B$

$\Delta \Vdash \mathcal{K}$ $\mapsto \Delta, u_1{:}A[\Phi], u_2{:}([u_1/x]_A B)[\Phi] \Vdash \mathcal{K}$
$u{:}(\Sigma x{:}A.B)[\Phi] \in \Delta$ active $+ \Phi \vdash u \leftarrow (u_1\,,\,u_2) : \Sigma x{:}A.\, B$

Flattening Σ-types

$\Delta \Vdash \mathcal{K}$ $(u{:}A[\Phi] \in \Delta$ active$)$ $\mapsto \Delta, v{:}([\sigma^{-1}]A)[\Phi'] \Vdash \mathcal{K} \;+\; \Phi \vdash u \leftarrow v[\sigma] : A$
$\Phi = \Phi_1, x : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, \Sigma z{:}B.\, C,\ \Phi_2$ $\Phi' = \Phi_1, x_1 : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, B,\ x_2 : \Pi \boldsymbol{y}{:}\boldsymbol{A}.\, [x_1\, \boldsymbol{y}/z]C,\ \Phi_2$
$\sigma^{-1} = [\lambda \boldsymbol{y}.\, (x_1\, \boldsymbol{y}\,, x_2\, \boldsymbol{y})/x]$ $\sigma = [\lambda \boldsymbol{y}.\, \mathsf{fst}\,(x\, \boldsymbol{y})/x_1, \lambda \boldsymbol{y}.\, \mathsf{snd}\,(x\, \boldsymbol{y})/x_2]$

Pruning

$\Delta \Vdash \mathcal{K}$ $\mapsto \Delta' \Vdash [\![\eta]\!]\mathcal{K}$
$(\Psi \vdash u[\rho] = M : C) \in \mathcal{K}$ if $\Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta'$; η and $\eta \neq \mathsf{id}$

Same meta-variable

$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = u[\xi] : C$ $\mapsto \Delta, v{:}A[\Phi_0] \Vdash \mathcal{K} + \Phi \vdash u \leftarrow v[\mathsf{id}_{\Phi_0}] : A$
$u{:}A[\Phi] \in \Delta$ if $\rho \cap \xi : \Phi \Rightarrow \Phi_0$

Failing occurs check

$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = M : C$ $\mapsto \bot$ if $\mathsf{FV}^{\mathsf{rig}}(M) \not\subseteq \rho$

$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = M : C$ $\mapsto \bot$ if $M = M'\{u[\xi]\}^{\mathsf{srig}} \neq u[\xi]$

Solving (with successful occurs check)

$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = M : C$ $\mapsto \Delta \Vdash \mathcal{K} + \Phi \vdash u \leftarrow M' : A$
$(u{:}A[\Phi]) \in \Delta;\ u \notin \mathsf{FMV}(M)$ if $M' = [\rho/\hat{\Phi}]^{-1}M$ exists

Fig. 5. Unification steps.

Flattening Σ-types $u : P[z : \Pi x{:}A.\, \Sigma y{:}B.\, C] \Vdash u[\lambda x.\, (z_1\, x\,,\, z_2\, x)] = M$
By splitting z into two functions z_1, z_2 we arrive at $u : P[z_1 : \Pi x{:}A.\, B,\ z_2 : \Pi x{:}A.\, [z_1\, x/y]C] \Vdash u[\lambda x.\, z_1\, x,\ \lambda x.\, z_2\, x] = M$ and continue with η-contraction.

Solving in spite of non-linearity $u[x, x, z] = \mathsf{suc}\, z$
The non-linear occurrence of x on the l.h.s. can be ignored since x is not free on the r.h.s. We can solve this constraint by by $u[x, y, z] = \mathsf{suc}\, z$.

Pruning $u[x] = \mathsf{suc}(v[x, y])$ and $v[x, \mathsf{zero}] = \mathsf{f}(x, \mathsf{zero})$
Since u depends only on x, necessarily v cannot depend on y. We can prune away the second parameter of v by setting $v[x, y] = v'[x]$. This turns the

second constraint into the pattern $v'[x] = \mathsf{f}(x, \mathsf{zero})$, yielding the solution $u[x] = \mathsf{suc}(\mathsf{f}(x, \mathsf{zero}))$.
Note that pruning is more difficult in case of meta-variable nesting. If instead $u[x] = \mathsf{suc}(v[x, w[y]])$ then there are two cases: either v does not depend on its second argument or w is constant. Pruning as we describe it in this article cannot be applied to this case; Reed [16] proceeds here by replacing y by a placeholder "_". Once w gets solved the placeholder might occur as argument to v, where it can be pruned. If the placeholder appears in a rigid position, the constraints have no solution.

Pruning and non-linearity $u[x, x] = v[x]$ and $u'[x, x] = v'[x, y]$
Even though we cannot solve for u due to the non-linear x, pruning x from v could lose solutions. However, we can prune y from v' since only x can occur in $v'[x, y]$.

Failing occurs check $u[x] = \mathsf{suc}\, y$
Pruning y fails because it occurs rigidly. The constraint set has no solution.

Same meta-variable $u[x, y, x, z] = u[x, y, y, x]$
Since variables x, y, z are placeholders for arbitrary *open* well-typed terms, of which infinitely many exists for every type, the above equation can only hold if u does not depend on its 3rd and 4th argument. Thus, we can solve by $u[x, y, z, x'] = v[x, y]$ where $[x, y]$ is the *intersection* of the two variable environments $[x, y, x, z]$ and $[x, y, y, x]$.

Recursive occurrence $u[x, y, x] = \mathsf{suc}\, u[x, y, y]$
Here, u has a *strong* rigid occurrence in its own definition. Even though not in the pattern fragment, this only has an infinite solution: consider the instance $u[z, z, z] = \mathsf{suc}\, u[z, z, z]$. Consequently, the occurs check signals unsolvability. [17, p. 105f] motivates why only *strong* rigid recursive occurrences force unsolvability. For instance, $f : \mathsf{nat} \to \mathsf{nat} \vdash u[f] = \mathsf{suc}\,(f\,(u[\lambda x.\, \mathsf{zero}]))$ has solution $u[f] = \mathsf{suc}\,(f\,(\mathsf{suc}\,\mathsf{zero}))$ in spite of a rigid occurrence of u in its definition.
If u occurs flexibly in its own definition, like in $u[x] = v[u[x]]$, we cannot proceed until we know more of v. Using the other constraints, we might manage to prune v's argument, arriving at $u[x] = v[]$, or find the solution of v directly; in these cases, we can revisit the constraint on u.

The examples suggest a *strategy* for implementation: Lowering can be integrated triggered by decomposition to resolve eliminations of a meta variable $E[u[\sigma]]$. After decomposition we have a set of $u[\sigma] = M$ problems. We try to turn the σs into variable substitutions by applying η-contraction, and where this gets stuck, elimination of projections and Σ-flattening. Solution of constraints $u[\rho] = M$ can then be attempted by pruning, where a failing occurs check signals unsolvability.

3.3 Inverting substitutions

A most general solution for a constraint $u[\sigma] = M$ can only be hoped for if σ is a variable substitution. For instance $u[\mathsf{true}] = \mathsf{true}$ admits already two different solutions $u[x] = x$ and $u[x] = \mathsf{true}$ that are pure λ-terms. In a language with computation such as Agda infinitely more solutions are possible, because $u[x]$ could be defined by cases on x and the value of $u[\mathsf{false}]$ is completely undetermined.

But even constraints $u[\rho] = M$ can be ambiguous if the variable substitution ρ is not linear, i. e., no bijective variable renaming. For example, $u[x, x] = x$ has solutions $u[x, y] = x$ and $u[x, y] = y$. Other examples, like $u[x, x, z] = z$, which has unique solution $u[x, y, z] = z$, suggest that we can ignore non-linear variable occurrences as long as they do not occur on the r.h.s. Indeed, if we define a variable substitution ρ to be *invertible for term M* if there is exactly one M' such that $[\rho]M' = M$, then linearity is a sufficient, but not necessary condition. However, it is necessary that ρ must be linear if restricted to the free variables of (β-normal!) M. Yet instead of computing the free variables of M, checking that ρ is invertible, inverting ρ and applying the result to M, we can directly try to invert the effect of the substitution ρ on M.

For a variable substitution $\Psi \vdash \rho \Leftarrow \Phi$ and a term/substitution $\alpha ::= M \mid R \mid \tau$ in context Ψ, we define the partial operation $[\rho/\hat{\Phi}]^{-1}\alpha$ by

$$
\begin{array}{lll}
[\rho/\hat{\Phi}]^{-1}x & = y & \text{if } x/y \in \rho/\hat{\Phi} \text{ and there is no } z \neq y \text{ with } x/z \in \rho/\hat{\Phi}, \\
 & & \text{undefined otherwise} \\
[\rho/\hat{\Phi}]^{-1}c & = c & \\
[\rho/\hat{\Phi}]^{-1}(u[\tau]) & = u[\tau'] & \text{where } \tau' = [\rho/\hat{\Phi}]^{-1}\tau
\end{array}
$$

and homeomorphic in all other cases by

$$
\begin{array}{lll}
[\rho/\hat{\Phi}]^{-1}(R\,M) & = R'\,M' & \text{where } R' = [\rho/\hat{\Phi}]^{-1}R \text{ and } M' = [\rho/\hat{\Phi}]^{-1}M \\
[\rho/\hat{\Phi}]^{-1}(\pi\,R) & = \pi\,R' & \text{where } R' = [\rho/\hat{\Phi}]^{-1}R \\
[\rho/\hat{\Phi}]^{-1}(\lambda x.\,M) & = \lambda x.\,M' & \text{if } x \text{ not declared or free in } \sigma \\
 & & \text{and } M' = [\rho, x/\hat{\Phi}, x]^{-1}M \\
[\rho/\hat{\Phi}]^{-1}(M\,,\,N) & = (M'\,,\,N') & \text{where } M' = [\rho/\hat{\Phi}]^{-1}M \text{ and } N' = [\rho/\hat{\Phi}]^{-1}N \\
[\rho/\hat{\Phi}]^{-1}(\cdot) & = \cdot & \\
[\rho/\hat{\Phi}]^{-1}(\tau, M) & = \tau', M' & \text{if } \tau' = [\rho/\hat{\Phi}]^{-1}\tau \text{ and } M' = [\rho/\hat{\Phi}]^{-1}M\ .
\end{array}
$$

We can show by induction on α, that inverse substitution $[\rho/\hat{\Phi}]^{-1}\alpha$ is correct, preserves well-typedness and commutes with meta substitutions.

Lemma 3 (Inverse and meta-substitution commute). *Let ρ be a variable substitution, and $\alpha ::= M \mid R \mid \tau$. If $[\rho/\hat{\Phi}]^{-1}\alpha$ and $[\rho/\hat{\Phi}]^{-1}([\![\theta]\!]\alpha)$ exist then $[\rho/\hat{\Phi}]^{-1}([\![\theta]\!]\alpha) = [\![\theta]\!]([\rho/\hat{\Phi}]^{-1}\alpha)$.*

Proof. By simultaneous induction on the structure of α. □

Lemma 4 (Soundness of inverse substitution). *If $[\rho/\hat{\Phi}]^{-1}\alpha$ exists then $[\rho]_{\Phi}([\rho/\hat{\Phi}]^{-1}\alpha) = \alpha$.*

Proof. By simultaneous induction on the structure of α. □

Lemma 5 (Completeness of inverse substitution). *If* $[\rho]_\Phi \alpha = \alpha'$ *and* $\rho \upharpoonright \mathsf{FV}(\alpha)$ *is linear then* $\alpha = [\rho/\hat{\Phi}]^{-1}\alpha'$ *exists.*

Proof. By simultaneous induction on the structure of α. □

Lemma 6 (Well-typedness of inverse substitution). *Let* $\Delta;\Psi \vdash \rho \Leftarrow \Phi$.

1. *If* $[\rho/\hat{\Phi}]^{-1}M$ *exists and* $\Delta;\Psi \vdash_{\mathcal{K}} M \Leftarrow [\rho]_\Phi A$ *then* $\Delta;\Phi \vdash_{\mathcal{K}} [\rho/\hat{\Phi}]^{-1}M \Leftarrow A$.
2. *If* $[\rho/\hat{\Phi}]^{-1}R$ *exists and* $\Delta;\Psi \vdash_{\mathcal{K}} R \Rightarrow [\rho]_\Phi A$ *then* $\Delta;\Phi \vdash_{\mathcal{K}} [\rho/\hat{\Phi}]^{-1}R \Rightarrow A$.
3. *If* $[\rho/\hat{\Phi}]^{-1}\tau$ *exists and* $\Delta;\Psi \vdash_{\mathcal{K}} \tau \Leftarrow \Psi_1$ *then* $\Delta;\Phi \vdash_{\mathcal{K}} [\rho/\hat{\Phi}]^{-1}\tau \Leftarrow \Psi_1$.

Proof. By simultaneous structural induction on the definition of inverse substitutions for τ, M, R. □

3.4 Pruning

If the constraint $u[\sigma] = M$ has a solution θ, then $[[\![\theta]\!]\sigma]\theta(u) = [\![\theta]\!]M$, and since θ is closed ($\mathsf{FMV}(\theta) = \emptyset$), we have $\mathsf{FV}(\sigma) \supseteq \mathsf{FV}([\![\theta]\!]M)$. Thus, if $\mathsf{FV}(M) \not\subseteq \mathsf{FV}(\sigma)$ we can try to find a most general meta-substitution η which *prunes* the free variables of M that are not in the range of σ, such that $\mathsf{FV}([\![\eta]\!]M) \subseteq \mathsf{FV}(\sigma)$. For instance, in case $u[x] = \mathsf{suc}\ v[x,y]$, the meta-substitution $x,y.\,v'[x]/v$ does the job. However, pruning fails either if pruned variables occur rigidly, like in $u[x] = \mathsf{c}\ y\ v[x,y]$ (constraint unsolvable), or if the flexible occurrence is under another meta variable, like in $u[x] = v[x, w[x,y]]$. Here, two minimal pruning substitutions $\eta_1 = x,y.\,v'[x]/v$ and $\eta_2 = x,y.\,w'[x]/w$ exist which are not instances of each other—applying pruning might lose solutions.

We restrict pruning to situations $u[\rho] = M$ where ρ is a variable substitution. This is because we view pruning as a preparatory step to inverting ρ on M—which only makes sense for variable substitutions. Also, we do not consider partial pruning, as in pruning y from v in the situation $u[x] = v[x,y,w[x,y]]$, obtaining $u[x] = v'[x, w[x,y]]$. Such extensions to pruning are thinkable, but we have no data indicating that they strengthen unification significantly in practice. Fig. 6 presents the rules for the judgements

$$\begin{array}{ll} \mathsf{prune\_ctx}_\rho(\tau \ / \ \Psi_1) \Rightarrow \Psi_2 & \text{prune } \tau \text{ such that } \mathsf{FV}^{\mathsf{rig}}([\tau]\mathsf{id}_{\Psi_2}) \subseteq \rho \\ \Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta';\ \eta & \text{prune } M \text{ such that } \mathsf{FV}([\![\eta]\!]M) \subseteq \rho. \end{array}$$

When pruning substitution τ with domain Ψ_1 we look at each term M in τ which substitutes for an $x{:}A$ of Ψ_1. If M has a rigid occurrence of a variable $y \notin \rho$, we discard the entry $x{:}A$ from the domain Ψ_1, thus, effectively removing M from τ. If M has no occurrence of such an y we keep $x{:}A$. However, since we might have removed prior entries from Ψ_1 we need to ensure A is still well-formed, by validating that its free variables are bound in the pruned context. Finally, if M has a flexible occurrence of a $y \notin \rho$, pruning fails. Examples:

$$\begin{array}{lllll} 1. & \mathsf{prune\_ctx}_x(\mathbf{c}\,x,\ y & /\ x'{:}A,\ y'{:}B) & \Rightarrow & x'{:}A \\ 2. & \mathsf{prune\_ctx}_y(\mathbf{c}\,x,\ y & /\ x'{:}A,\ y'{:}B) & \Rightarrow & y'{:}B \\ 3. & \mathsf{prune\_ctx}_y(\mathbf{c}\,x,\ u[y] & /\ x'{:}A,\ y'{:}B) & \Rightarrow & y'{:}B \\ 4. & \mathsf{prune\_ctx}_y(u[x],\ y & /\ x'{:}A,\ y'{:}B) & & \text{fails} \end{array}$$

$\boxed{\mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2}$ Prune $\tau : \Psi_1$, returning a sub-context Ψ_2 of Ψ_1.

$$\frac{}{\mathsf{prune\_ctx}_\rho(\cdot \;/\; \cdot) \Rightarrow \cdot} \qquad \frac{\mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2 \quad \mathsf{FV}^{\mathsf{rig}}(M) \not\subseteq \rho}{\mathsf{prune\_ctx}_\rho(\tau, M \;/\; \Psi_1, x{:}A) \Rightarrow \Psi_2}$$

$$\frac{\mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2 \quad \mathsf{FV}(M) \subseteq \rho \quad \mathsf{FV}(A) \subseteq \hat{\Psi}_2}{\mathsf{prune\_ctx}_\rho(\tau, M \;/\; \Psi_1, x{:}A) \Rightarrow \Psi_2, x{:}A}$$

$\boxed{\Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta';\; \eta}$ Prune M, returning $\Delta' \vdash \eta \Leftarrow \Delta$.

$$\frac{v{:}B[\Psi_1] \in \Delta \quad \mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2 \quad \Psi_2 \neq \Psi_1 \quad \mathsf{FV}(B) \subseteq \hat{\Psi}_2 \quad \eta = \hat{\Psi}_1.v'/v}{\Delta \vdash \mathsf{prune}_\rho(v[\tau]) \Rightarrow [\![\eta]\!](\Delta, v'{:}B[\Psi_2]);\; \eta}$$

$$\frac{v{:}B[\Psi_1] \in \Delta \quad \mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_1}{\Delta \vdash \mathsf{prune}_\rho(v[\tau]) \Rightarrow \Delta;\; \mathsf{id}_\Delta} \qquad \frac{x \in \rho}{\Delta \vdash \mathsf{prune}_\rho\, x \Rightarrow \Delta;\; \mathsf{id}_\Delta}$$

$$\frac{}{\Delta \vdash \mathsf{prune}_\rho\, c \Rightarrow \Delta;\; \mathsf{id}_\Delta} \qquad \frac{\Delta \vdash \mathsf{prune}_\rho\, R \Rightarrow \Delta_1;\; \eta_1 \quad \Delta_1 \vdash \mathsf{prune}_\rho([\![\eta_1]\!]M) \Rightarrow \Delta_2;\; \eta_2}{\Delta \vdash \mathsf{prune}_\rho(R\, M) \Rightarrow \Delta_2;\; [\![\eta_2]\!]\eta_1}$$

$$\frac{\Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta';\; \eta}{\Delta \vdash \mathsf{prune}_\rho(\pi\, M) \Rightarrow \Delta';\; \eta} \qquad \frac{\Delta \vdash \mathsf{prune}_{\rho,x}\, M \Rightarrow \Delta';\; \eta}{\Delta \vdash \mathsf{prune}_\rho(\lambda x.\, M) \Rightarrow \Delta';\; \eta}$$

$$\frac{\Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta_1;\; \eta_1 \quad \Delta_1 \vdash \mathsf{prune}_\rho([\![\eta_1]\!]N) \Rightarrow \Delta_2;\; \eta_2}{\Delta \vdash \mathsf{prune}_\rho\, (M\,,\, N) \Rightarrow \Delta_2;\; [\![\eta_2]\!]\eta_1}$$

Fig. 6. Pruning.

Pruning a term M with respect to ρ ensures that all rigid variables of M are in the range of ρ (see variable rule). Also, for each rigid occurrence of a meta variable $v[\tau]$ in M we try to prune the substitution τ. If τ is already pruned, we leave v alone; otherwise, if the domain Ψ_1 of τ shrinks to Ψ_2 then we replace $v : B[\Psi_1]$ by a new meta-variable $v' : B[\Psi_2]$ with domain Ψ_2. However, we need to ensure that the type B still makes sense in Ψ_2; otherwise, pruning fails.

Lemma 7 (Soundness and completeness of pruning).

1. *If* $\Delta \vdash_{\mathcal{K}} \Psi_1\,\mathsf{ctx}$ *and* $\mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2$ *then* $\Delta \vdash_{\mathcal{K}} \Psi_2\,\mathsf{ctx}$ *and* $\mathsf{FV}([\tau]\mathsf{id}_{\Psi_2}) \subseteq \rho$. *Additionally, if* $x \in \Psi_1 \setminus \Psi_2$ *then* $\mathsf{FV}^{\mathsf{rig}}([\tau]x) \not\subseteq \rho$.
2. *If* $\Delta \vdash \mathsf{prune}_\rho\, M \Rightarrow \Delta';\; \eta$ *then* $\Delta' \vdash_{\mathcal{K}} \eta \Leftarrow \Delta$ *and* $\mathsf{FV}([\![\eta]\!]M) \subseteq \rho$. *Also, if* θ *solves* $\Psi \vdash u[\rho] = M_0\{M\}^{\mathsf{rig}} : C$ *then there is some* θ' *such that* $\theta = [\![\theta']\!]\eta$.

Proof. Each by induction on the pruning derivation.

We detail 2., existence of θ': Since θ is a solution of the constraint, $[\![\theta]\!](u[\rho]) = [[\![\theta]\!]\rho](\theta(u)) = [\rho](\theta(u)) = [\![\theta]\!]M_0$, in particular $\mathsf{FV}(\rho) = \rho \supseteq \mathsf{FV}([\![\theta]\!]M_0)$. This entails $\mathsf{FV}([\![\theta]\!]M) \subseteq \rho$.

Consider the interesting case $M = v[\tau]$ with $\mathsf{prune\_ctx}_\rho(\tau \;/\; \Psi_1) \Rightarrow \Psi_2$ and $\eta = \hat{\Psi}_1.v'[\mathsf{id}_{\Psi_2}]/v$. Let $N = \theta(v)$. We have $\mathsf{FV}([\llbracket\theta\rrbracket\tau]N) \subseteq \rho$. If we can show $\mathsf{FV}(N) \subseteq \hat{\Psi}_2$, then we can finish by setting $\theta' = \theta, \hat{\Psi}_2.N/v'$.

Assume now some $x \in \mathsf{FV}(N)$ with $x \notin \hat{\Psi}_2$. By 1., $\mathsf{FV}^{\mathsf{rig}}([\tau]x) \not\subseteq \rho$, which entails that $\mathsf{FV}([\llbracket\theta\rrbracket\tau]x) \not\subseteq \rho$. This is in contradiction to $\mathsf{FV}([\llbracket\theta\rrbracket\tau]N) \subseteq \rho$. □

In an implementation, we may combine pruning with inverse substitution and the occurs check. Since we already traverse the term M for pruning, we may also check whether $[\rho/\hat{\Phi}]^{-1}M$ exists and whether u occurs in M.

3.5 Unifying two identical existential variables

Any solution $\hat{\Phi}.N/u$ for a meta variable $u : A[\Phi]$ with constraint $u[\rho] = u[\xi]$ must fulfill $[\rho]N = [\xi]N$, which means that $[\rho]x = [\xi]x$ for all $x \in \mathsf{FV}(N)$. This means that u can only depend on those of its variables in Φ that are mapped to the same term by ρ and ξ. Thus, we can substitute u by $\hat{\Phi}.v[\rho']$ where ρ' is the *intersection* of substitutions ρ and ξ. Similarly to context pruning, we obtain ρ' as $[\rho]\mathsf{id}_{\Phi'}$, which identical to $[\xi]\mathsf{id}_{\Phi'}$, where Φ' is a subcontext of Φ mentioning only the variables that have a common image under ρ and ξ. This process is given as judgement $\boxed{\rho \cap \xi : \Phi \Rightarrow \Phi'}$ with the following rules:

$$\frac{}{\cdot \cap \cdot : \cdot \Rightarrow \cdot}$$

$$\frac{\rho \cap \xi : \Phi \Rightarrow \Phi'}{(\rho, y) \cap (\xi, y) : (\Phi, x{:}A) \Rightarrow (\Phi', x{:}A)} \qquad \frac{\rho \cap \xi : \Phi \Rightarrow \Phi' \quad z \neq y}{(\rho, z) \cap (\xi, y) : (\Phi, x{:}A) \Rightarrow \Phi'}$$

Lemma 8 (Soundness of intersection). *If* $\Delta; \Psi \vdash_{\mathcal{K}} \rho, \xi \Leftarrow \Phi$ *and* $\rho \cap \xi : \Phi \Rightarrow \Phi'$, *then* $\Delta \vdash_{\mathcal{K}} \Phi' \;\mathsf{ctx}$ *and* $\Delta; \Phi \vdash_{\mathcal{K}} \mathsf{id}_{\Phi'} \Leftarrow \Phi'$ *and* $[\rho]\mathsf{id}_{\Phi'} = [\xi]\mathsf{id}_{\Phi'}$.

Proof. Structural induction on the first derivation. We consider the interesting case, where we actually retain a declaration because the variable y in ρ and ξ is *strictly shared*, meaning $y = [\rho]x = [\xi]x$ for some x.

$$\frac{\rho \cap \xi : \Phi \Rightarrow \Phi'}{(\rho, y) \cap (\xi, y) : (\Phi, x{:}B) \Rightarrow \Phi', x{:}B}$$

$\Delta; \Psi \vdash_{\mathcal{K}} (\rho, y) \Leftarrow (\Phi, x{:}B)$ by assumption
$\Delta; \Psi \vdash_{\mathcal{K}} \rho \Leftarrow \Phi$ by inversion
$\Psi(y) = B' =_{\mathcal{K}} [\rho](B)$ by inversion
$\Delta; \Psi \vdash_{\mathcal{K}} (\xi, y) \Leftarrow (\Phi, x{:}B)$ by assumption
$\Delta; \Psi \vdash_{\mathcal{K}} \xi \Leftarrow \Phi$ by inversion
$\Psi(y) = B' =_{\mathcal{K}} [\xi](B)$ by inversion
$B' =_{\mathcal{K}} [\rho](B) =_{\mathcal{K}} [\xi](B)$ by previous lines
recall that ρ and ξ are variable substitutions (more importantly $\mathsf{FMV}(\rho) = \mathsf{FMV}(\xi) = \emptyset$ and hence they do not interact with the constraints $\mathcal{K}$)
B can only depend on variables strictly shared between ρ and ξ
$\Delta \vdash_{\mathcal{K}} \Phi' \;\mathsf{ctx}$ by i.h.

$\Delta; \Phi' \vdash_{\mathcal{K}} B \Leftarrow \mathsf{type}$ since Φ' contains exactly those variables strictly shared by ρ and ξ
$\Delta \vdash_{\mathcal{K}} (\Phi', x{:}B)\ \mathsf{ctx}$. □

Let us now reconsider the rule "Same meta-variable". We have shown that the resulting Φ' of computing the intersection of ρ and ξ, is indeed well-formed (Lemma 8). We can also justify why in the unification rule itself the type A of two existential variables must be well-typed in the pruned context Φ'. Recall that by typing invariant, we know that $\Delta; \Psi \vdash_{\mathcal{K}} \rho \Leftarrow \Phi$ and $\Delta; \Psi \vdash_{\mathcal{K}} \xi \Leftarrow \Phi$ and $[\rho]A =_{\mathcal{K}} [\xi]A$. But this means that A can only depend on the variables mapped to the same term by ρ and ξ. Since Φ_0 is exactly the context, which captures those shared variables, A must also be well-typed in Φ_0 modulo $\mathcal{K}$. Although we have restricted intersection to variable substitutions, it could be extended to meta-ground substitutions, i.e., substitutions that do not contain any meta-variables.

4 Correctness

Theorem 1 (Termination). *The algorithm terminates and results in one of the following states:*

- *A solved sate where only assignments $\Psi \vdash u \leftarrow M : A$ remain.*
- *A stuck state, i.e., no transition rule applies.*
- *Failure $\bot$.*

Proof. The size $|M|$ of a term be as usual the number of nodes and leaves in its tree representation, with the exception that we count λ-nodes twice. This modification has the effect that $|\lambda x M| + |R| > |M| + |R\,x|$, hence, an η-expanding decomposition step also decreases the sum of the sizes of the involved terms [6]. We define the size $|A[\Phi]|$ of a type A in context Φ by $|P[\Phi]| = 1 + \sum_{A \in \Phi} |A[]|$, $|(\Pi x{:}A.\, B)[\Phi]| = 1 + |B[\Phi, x{:}A]|$ and $|(\Sigma x{:}A.\, B)[\Phi]| = 1 + |A[\Phi]| + |B[\Phi]|$. The size of a type can then be obtained as $|A| = |A[]|$ and the size of a context as $|\Phi| = \sum_{A \in \Phi} |A|$. The purpose of this measure is to give Σ-types a large weight that can "pay" for *flattening*.

The weight of a solved constraint be 0, whereas the weight $|K|$ for a constraint $\Psi \vdash M = M' : C$ be the ordinal $(|M| + |M'|)\omega + |\Psi|$ if a decomposition step can be applied, and simply $|\Psi|$ else. Similarly, the weight of constraint $\Phi \mid R{:}A \vdash E = E'$ be $(|E| + |E'|)\omega + |\Psi|$. Finally, the weight $|\Delta \Vdash \mathcal{K}|$ of a unification problem be the ordinal

$$\sum_{u:A[\Phi] \in \Delta \text{ active}} |A[\Phi]|\omega^2 + \sum_{K \in \mathcal{K}} |K|.$$

By inspection of the transition rules we can show that each unification step reduces the weight of the unification problem. □

4.1 Solutions to unification

A solution to a set of equations $\mathcal{K}$ is a meta-substitution θ for all the meta-variables in Δ s.t. $\Delta' \vdash \theta \Leftarrow \Delta$ and

1. for every $\Psi \vdash u \leftarrow M : A$ in $\mathcal{K}$ we have $\hat{\Psi}.M/u \in \theta$,
2. for all equations $\Psi \vdash M = N : A$ in $\mathcal{K}$, we have $[\![\theta]\!]M = [\![\theta]\!]N$.

A *ground* solution to a set of equations $\mathcal{K}$ can be obtained from a solution to $\mathcal{K}$ by applying a grounding meta-substitution θ' where $\cdot \vdash \theta' \Leftarrow \Delta'$ to the solution θ. We write $\theta \in \mathsf{Sol}(\Delta \Vdash \mathcal{K})$ for a ground solution to the constraints $\mathcal{K}$.

First, we show that the pruning rule preserves solutions; this is a crucial lemma towards showing that the unification rules preserve solutions.

Lemma 9 (Pruning preserves solutions).
Let $\theta \in \mathsf{Sol}(\Delta \Vdash \mathcal{K})$ where $\cdot \vdash \theta \Leftarrow \Delta$ and $(\Psi \vdash u[\rho] = M\{M'\} : A) \in \mathcal{K}$. If $\Delta \vdash \mathsf{prune}_\rho\, M' \Rightarrow \Delta_p;\ \theta_p$ then there exists a meta-substitution θ' s.t. $\cdot \vdash \theta' \Leftarrow \Delta_p$ and $[\![\theta']\!](\theta_p) = \theta$.

Proof. Induction on the derivation $\mathcal{D}$ of the pruning judgment. We write $M\{M'\}$ to indicate that M' occurs as a subterm in M. We consider the interesting case here where we actually prune away some bound variables.

Case $\Delta; \Psi \vdash v[\tau] \Rightarrow C$ and

$$\mathcal{D} = \frac{\mathsf{prune\_ctx}_\rho(\tau \ /\ \Psi_1) \Rightarrow \Psi_2 \quad \Delta' = [\![\hat{\Psi}_1.v'[\mathsf{id}_{\Psi_2}]/v]\!](\Delta, v'{:}Q[\Psi_2])}{\Delta \vdash \mathsf{prune}_\rho(v[\tau]) \Rightarrow \Delta';\ \hat{\Psi}_1.v'/v}$$

where $v{:}B[\Psi_1] \in \Delta$ and $\mathsf{FV}(B) \subseteq \hat{\Psi}_2$ by assumption
$\cdot \vdash \theta \Leftarrow \Delta$ by assumption
$\hat{\Psi}_1.N/v \in \theta$ and $\cdot; [\![\theta]\!]\Psi_1 \vdash_{\mathcal{K}} N \Leftarrow [\![\theta]\!]B$ by previous lines
We also must have : $u{:}A'[\Phi'] \in \Delta$ and $\hat{\Phi}'.M_0/u \in \theta$. by assumption
$[\![\theta]\!](u[\rho]) = [\![\theta]\!]M$ because θ is a solution
$[[\![\theta]\!]\rho]M_0 = [\rho]M_0 = [\![\theta]\!]M$ by previous lines
$\mathsf{FV}([\rho]M_0) = \mathsf{FV}(\rho) = \mathsf{FV}([\![\theta]\!]M) \supseteq \mathsf{FV}([\![\theta]\!](v[\tau])) = \mathsf{FV}([[\![\theta]\!]\tau]N)$
$\mathsf{FV}([[\![\theta]\!]\tau]N) \subseteq \mathsf{FV}(\rho)$ by previous lines
Let's assume there is an $x \in \mathsf{FV}(N)$ s.t. $x \in \Psi_1$ but $x \notin \Psi_2$. We note that x must be occur in a rigid position, i.e. not in the delayed substitution which is associated with a meta-variable occurring in τ; otherwise $\mathsf{prune\_ctx}_\rho(\tau \ /\ \Psi_1)$ would not succeed. Hence $x \in \mathsf{FV}([\![\theta]\!]\tau)$. But this means $x \in \mathsf{FV}(\rho)$ which contradicts $\mathsf{FV}([[\![\theta]\!]\tau]N) \subseteq \mathsf{FV}(\rho)$.

By completeness of pruning, we know $\mathsf{prune\_ctx}$ generates the maximal Ψ_2, and hence, N can at most depend on the variables in Ψ_2.

We need to show that there exists a θ' s.t. $\cdot \vdash_{\mathcal{K}} \theta' \Leftarrow \Delta'$ and $[\![\theta']\!]\theta_p = \theta$.
Recall, $\mathsf{FV}(B) \subseteq \hat{\Psi}_2$ and hence $\cdot; [\![\theta]\!]\Psi_2 \vdash_{\mathcal{K}} [\![\theta]\!]B \Leftarrow \mathsf{type}$;
moreover, $\cdot; [\![\theta]\!]\Psi_2 \vdash_{\mathcal{K}} N \Leftarrow [\![\theta]\!]B$. Let $\theta = \theta_1, \hat{\Psi}_1.N/v, \theta_2$. Then there exists a $\theta' = \theta, \hat{\Psi}_2.N/v'$ s.t. $[\![\theta']\!](\hat{\Psi}_1.v'/v) = \theta$. □

Next, we prove that transitions preserve solutions. We first observe that if we start in a state $\Delta_0 \Vdash K_0$ and transition to a state $\Delta_1 \Vdash K_1$ the meta-variable context strictly grows, i.e., $\mathsf{dom}(\Delta_0) \subseteq \mathsf{dom}(\Delta_1)$. We subsequently show that if we have a solution for $\Delta_0 \Vdash K_0$, then transitioning to a new state $\Delta_1 \Vdash K_1$ will not add any additional solutions nor will it destroy some solution we may already have. In other words, any additional constraints which may be added in $\Delta_1 \Vdash K_1$ are consistent with the already existing solution.

Theorem 2 (Transitions preserve solutions). *Let* $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$.

1. *If* $\theta_0 \in \mathsf{Sol}(\Delta_0 \Vdash \mathcal{K}_0)$ *then there exists a meta-substitution* θ' *s.t.* $\Delta_1 \vdash \theta' \Leftarrow \Delta_0$ *and a solution* $\theta_1 \in \mathsf{Sol}(\Delta_1 \Vdash \mathcal{K}_1)$ *such that* $[\![\theta_1]\!]\theta' = \theta_0$.
2. *If* $\theta_1 \in \mathsf{Sol}(\Delta_1 \Vdash \mathcal{K}_1)$ *then* $[\![\theta_1]\!]\mathsf{id}_{\Delta_0} \in \mathsf{Sol}(\Delta_0 \Vdash \mathcal{K}_0)$.

Proof. Proof by case analysis on the transitions. We only show the cases to prove that we are forward closed (statement 1).

Case: Decomposition Since Δ_0 does not change in any of the decomposition rules, the solution is almost trivially preserved; for the η-contraction rules, we simply observe that equality is always modulo η. For the **Eliminating projections** transition, we use the substitution lemma and observe that meta-substitutions and ordinary substitutions commute.

Case: Lowering If θ_0 is a solution for $\mathcal{K}_0$ and the active meta-variable $u{:}(\Pi x{:}A.B)[\Phi]$, then $\hat{\Phi}.M/u \in \theta_0$ and $\cdot; [\![\theta_0]\!]\Phi \vdash M \Leftarrow [\![\theta_0]\!](\Pi x{:}A.B)$. By inversion lemma, $M = \lambda x.N$ and taking into account the definition of meta-substitutions, we have $\cdot; [\![\theta_0]\!](\Phi, x{:}A) \vdash N \Leftarrow [\![\theta_0]\!]B$ and hence $\hat{\Phi}, x.N/v$ is a solution for v. Choose for $\theta' = \mathsf{id}_{\Delta_0}$ and for $\theta_1 = \theta, \hat{\Phi}, x.N/v$. The case for lowering Σ-types is similar.

Case: Flattening Using the substitution lemma, solutions are preserved.

Case: Pruning θ_0 is a solution for $\mathcal{K}_0$ and the active meta-variable $u{:}A[\Phi] \in \Delta_0$. Hence, $\hat{\Phi}.M/u \in \theta_0$. Moreover, if we have the constraint $\Psi \vdash u[\rho] = N : B$, we have $[\rho]M = [\![\theta_0]\!]N$. By previous soundness lemma for pruning, $\Delta_p \vdash \theta_p \Leftarrow \Delta_0$ and there exists a θ' s.t. $[\![\theta']\!]\theta_p = \theta$.

Case: Same meta-variable θ_0 is a solution for $\mathcal{K}_0$ and the active meta-variable $u{:}A[\Phi] \in \Delta_0$. Hence, $\hat{\Phi}.M/u \in \theta$ and $\cdot; [\![\theta_0]\!](\Phi) \vdash M \Leftarrow [\![\theta_0]\!]B$ where $B = [\rho]A$. Moreover, $[\rho]M = [\xi]M$. Therefore, $\mathsf{FV}([\rho]M) = \mathsf{FV}([\xi]M)$ and Φ_0 contains exactly those meta-variables which are shared among ρ and ξ by definition of $\rho \cap \xi$; hence $\mathsf{FV}(M) = \Phi_0$ and $\cdot; [\![\theta_0]\!](\Phi_0) \vdash M \Leftarrow [\![\theta_0]\!]B$; choosing id_{Δ_0} for θ' and for $\theta_1 = \theta, \hat{\Phi}_0.M/v$ solutions are preserved.

Case:Solving θ_0 is a solution for $\mathcal{K}_0$ and the active meta-variable $u{:}A[\Phi] \in \Delta_0$. Hence, $\hat{\Phi}.N/u \in \theta_0$. Therefore, we have $[\rho]N = [\![\theta]\!]M$. By completeness of inverse substitution, we know $N = [\rho/\Phi]^{-1}([\![\theta]\!]M)$. By assumption we also know $[\rho/\Phi]^{-1}M = M'$ exists. Therefore, by lemma that inverse and meta-substitution commute, we have $N = [\![\theta]\!]([\rho/\Phi]^{-1}M) = [\![\theta]\!]M'$. Therefore, the solution θ_0 is preserved. ☐

4.2 Transitions preserve types

Our goal is to prove that if we start with a well-typed unification problem our transitions preserve the type, i.e., we can never reach an ill-typed state and hence, we cannot generate a solution which may contain an ill-typed term.

Lemma 10 (Equality modulo is preserved by transitions).
If $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ *and* $A =_{\mathcal{K}_0} B$*, then* $A =_{\mathcal{K}_1} B$*.*

Proof. Let θ be a solution for $\mathcal{K}_0$; by assumption, we have that $[\![\theta]\!]A = [\![\theta]\!]B$. By theorem 2, transitions preserve solutions, we know θ is also a solution for K_1, and therefore $A =_{\mathcal{K}_1} B$. □

In the statement below it is again important to note that the meta-context strictly grows, i.e., $\Delta_0 \subseteq \Delta_1$. Therefore if Ψ, M, and A are well-formed with respect to Δ_0, they will be well-formed with respect to Δ_1.

Lemma 11 (Transitions preserve typing). *Let* $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$*.*

1. *If* $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow A$ *then* $\Delta_1; \Psi \vdash_{\mathcal{K}_1} M \Leftarrow A$*.*
2. *If* $\Delta_0; \Psi \vdash_{\mathcal{K}_0} R \Rightarrow A$ *then* $\Delta_1; \Psi \vdash_{\mathcal{K}_1} R \Rightarrow A'$ *and* $A =_{\mathcal{K}_1} A'$*.*

Proof. By induction on the derivation of $\Delta_0; \Psi \vdash_{\mathcal{K}_0} J$. Most cases are by inversion, appeal to induction hypothesis, and re-assembling the result. The most interesting case is transitioning between normal and neutral terms. Here we use the previous lemma on "Equality modulo preserved by transitions". □

Next, we define when a set of equations which constitute a unification problem are well-formed using the judgment $\Delta_0 \Vdash_{\mathcal{K}_0} \mathcal{K}\ \mathsf{wf}$, which states that each equation $\Psi \vdash M = N : A$ must be well-typed modulo the equations in $\mathcal{K}_0$, i.e., $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} N \Leftarrow A$. We simply write $\Delta_0 \Vdash \mathcal{K}\ \mathsf{wf}$ to mean $\Delta_0 \Vdash_{\mathcal{K}} \mathcal{K}\ \mathsf{wf}$.

Lemma 12 (Equations remain well-formed under meta-substitutions).
If $\Delta_0 \Vdash \mathcal{K}\ \mathsf{wf}$ *and* $\Delta_1 \vdash_{\mathcal{K}} \theta \Leftarrow \Delta_0$ *then* $\Delta_1 \Vdash [\![\theta]\!]\mathcal{K}\ \mathsf{wf}$*.*

Proof. By assumption $\Delta_0 \Vdash \mathcal{K}\ \mathsf{wf}$. By definition, for every constraint $\Psi \vdash M = N : A \in \mathcal{K}$, we have $\Delta_0; \Psi \vdash_{\mathcal{K}} M \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}} N \Leftarrow A$. By meta-substitution principle modulo (lemma 2), we know $\Delta_1; [\![\theta]\!]\Psi \vdash \mathcal{K}[\![\theta]\!]M \Leftarrow [\![\theta]\!]A$ and $\Delta_1; [\![\theta]\!]\Psi \vdash \mathcal{K}[\![\theta]\!]N \Leftarrow [\![\theta]\!]A$, and hence $\Delta_1 \Vdash [\![\theta]\!]K\ \mathsf{wf}$. □

Lemma 13 (Well-formedness of equations is preserved by transitions).
If $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ *and* $\Delta_0 \Vdash_{\mathcal{K}_0} \mathcal{K}\ \mathsf{wf}$ *then* $\Delta_1 \vdash_{\mathcal{K}_1} \mathcal{K}\ \mathsf{wf}$*.*

Proof. By assumption $\Delta \Vdash_{\mathcal{K}_0} \mathcal{K}\ \mathsf{wf}$, we know that for each $\Psi \vdash M = N : A \in \mathcal{K}$, $\Delta; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow A$ and $\Delta; \Psi \vdash_{\mathcal{K}_0} N \Leftarrow A$. By lemma 11, typing is preserved by transitions, we know that $\Delta; \Psi \vdash_{\mathcal{K}_1} M \Leftarrow A$ and $\Delta; \Psi \vdash_{\mathcal{K}_1} N \Leftarrow A$. Therefore $\Delta \Vdash_{\mathcal{K}_1} \mathcal{K}\ \mathsf{wf}$. □

Theorem 3 (Unification preserves types).
If $\Delta_0 \Vdash \mathcal{K}_0\ \mathsf{wf}$ *and* $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ *then* $\Delta_1 \Vdash \mathcal{K}_1\ \mathsf{wf}$*.*

Proof. By case analysis on the transition rules.

Case: Decomposition rules We consider the decomposition rule for pairs. Let $\mathcal{K}_0$ be the set of equations which contains $\Psi \vdash (M_1, M_2) = (N_1, N_2) : \Sigma x{:}A.B$. By assumption we have $\Delta_0; \Psi \vdash_{\mathcal{K}_0} (M_1, M_2) \Leftarrow \Sigma x{:}A.B$. By inversion, we have $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M_1 \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M_2 \Leftarrow [M_1/x]_A(B)$. By assumption we have $\Delta_0; \Psi \vdash_{\mathcal{K}_0} (N_1, N_2) \Leftarrow \Sigma x{:}A.B$. By inversion, we have $\Delta_0; \Psi \vdash_{\mathcal{K}_0} N_1 \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} N_2 \Leftarrow [N_1/x]_A(B)$. Let $K_1 = \mathcal{K}_0 \wedge \Psi \vdash M_1 = N_1 : A$. Then $\Delta_0 \Vdash K_1$ wf. Moreover, $\Delta_0; \Psi \vdash_{K_1} N_2 \Leftarrow [M_2/x]_A(B)$. Hence, $\Psi \vdash M_2 = N_2 : [M_1/x]_A(B)$ is well-formed and $\Delta_0 \Vdash K_2$ wf where we replace the constraint $\Psi \vdash (M_1, M_2) : \Sigma x{:}A.B$ with $\Psi \vdash M_1 = N_1 : A \wedge \Psi \vdash M_2 = N_2 : [M_1/x]_A(B)$ in $\mathcal{K}$.

Next, we consider the decomposition rules for evaluation contexts. Let $\mathcal{K}_0$ be the set of equations containing $\Psi \mid R : \Pi x{:}A.\, B \vdash E[\bullet\, M] = E'[\bullet\, M']$. By assumption this constraint is well-typed, and hence $\Delta_0; \Psi \vdash_{\mathcal{K}_0} R \Rightarrow \Pi x{:}A.\, B$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} R\, M \Rightarrow B_2$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} R\, M' \Rightarrow B_1$ where $B_1 =_{\mathcal{K}_0} B_2$. Let $\mathcal{K}_1 = \mathcal{K}_0 \wedge \Psi \vdash M = M' : A$. Clearly, $\Delta_0 \Vdash \mathcal{K}_1$ wf. Moreover, since the evaluation $E[R\, M]$ and $E'[R\, M']$ are well-typed modulo $\mathcal{K}_0$, we have also that $E'[R\, M]$ is well-typed modulo $\mathcal{K}_1$ and $\Psi \mid R\, M : [M/x]B \vdash E = E'$ is well-typed in Δ_0 modulo $\mathcal{K}_1$. Therefore, we have $\Delta_0 \Vdash \mathcal{K}_2$ wf where $\mathcal{K}_2 = \mathcal{K}_1 \wedge \Psi \mid R\, M : [M/x]B \vdash E = E'$.

Case:Pruning rule Let $\mathcal{K}_0$ be the set of equations containing $\Psi \vdash u[\rho] = M : A$. By assumption, we know that $\Delta_0; \Psi \vdash_{\mathcal{K}_0} u[\rho] \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow A$.

By soundness of pruning (lemma 7), we know that $\Delta_1 \vdash_{\mathcal{K}_0} \eta \Leftarrow \Delta_0$. By lemma 12, we know that $\Delta_1 \Vdash [\![\eta]\!](\mathcal{K}_0)$ wf

Case:Intersections Let $\mathcal{K}_0$ be the set of equations containing $\Psi \vdash u[\rho] = u[\xi] : C$. By assumption, we know that $\Delta_0; \Psi \vdash_{\mathcal{K}_0} u[\rho] \Leftarrow C$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} u[\xi] \Leftarrow C$. Let $u : A[\Phi] \in \Delta$. By inversion, we have $[\xi]A =_{\mathcal{K}_0} C =_{\mathcal{K}_0} [\rho]A$. This means $\mathsf{FV}([\xi]A) = \mathsf{FV}([\rho]A)$ and by definition of $\rho \cap \xi : \Phi \Rightarrow \Phi_0$, the context Φ_0 will contain exactly those variables shared in ξ and ρ. By soundness lemma 8, we have $\Delta_0 \vdash_{\mathcal{K}_0} \Phi_0$ ctx. Therefore, $\Delta_0; \Phi_0 \vdash_{\mathcal{K}_0} A \Leftarrow \mathsf{type}$ and $(\Delta_0, v{:}A[\Phi_0])$ mctx. By typing rules, we have $(\Delta_0, v{:}A[\Phi_0]); \Phi \vdash_{\mathcal{K}_0} u \Leftarrow A$ and $(\Delta_0, v{:}A[\Phi_0]); \Phi \vdash_{\mathcal{K}_0} v[\mathsf{id}_{\Phi_0}] \Leftarrow A$. Hence, $\Phi \vdash u \leftarrow v[\mathsf{id}_{\Phi_0} : A$ is well-typed in Δ_0 modulo $\mathcal{K}_0$. Hence, $\theta = \hat{\Phi}.v[\mathsf{id}_{\Phi_0}]/u$ is a well-formed meta-substitution. By lemma 2, we have $[\![\theta]\!]\Delta \Vdash ([\![\theta]\!]\mathcal{K}_0 \wedge [\![\theta]\!]\Phi \vdash u \leftarrow M : [\![\theta]\!]A)$ wf

Case:Solving Let $\mathcal{K}_0$ be the set of equations containing $\Psi \vdash u[\rho] = M : C$. By assumption $M' = [\rho/\hat{\Phi}]^{-1}M$ exists and $u{:}A[\Phi] \in \Delta$. Since $\Delta_0 \Vdash \mathcal{K}_0$ wf, we also have $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow C$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} u[\rho] \Leftarrow C$. By inversion, we have $C =_{\mathcal{K}_0} [\rho]A$. By lemma about the well-typedness of inverse substitutions (lemma 6), we have $\Delta_0; \Phi \vdash_{\mathcal{K}_0} M' \Leftarrow A$. Hence, $\theta = \hat{\Phi}.M'/u$ is a well-formed meta-substitution and by lemma 2, we have $[\![\theta]\!]\Delta_0 \Vdash [\![\theta]\!](\mathcal{K}_0 \wedge \Phi \vdash u \leftarrow M' : A)$ wf

□

5 Conclusion

We have presented a constraint-based unification algorithm which solves higher-order patterns dynamically and showed its correctness. There are several key aspects of our algorithm: First, we define pruning formally and show soundness in the dependently typed case. Our pruning operation differs from previous formulations in how it treats non-patterns which may occur in the term to be pruned: if it encounters a non-pattern term M where $\mathsf{FV}(M) \subseteq \rho$, then pruning may succeed; otherwise it fails. This strategy avoids non-termination problems present in previous formulations [2], but is also less ambitious than the algorithm proposed by Reed [16]. We have extended higher-order pattern unification to handle Σ-types; this has been an open problem so far. In practice, this is important in systems such as Agda where Σ-types are commonly used, but unification cannot handle Σ-types effectively. In systems such as Beluga, Twelf or Delphin, a limited form of Σ-types arises due to the world or context declaration. To be able to state that multiple assumptions are introduced at the same time, these systems employ Σ-types. In Beluga, we have implemented the flattening of Σ-types and it works well in type reconstruction.

Acknowledgments. We thank Jason Reed for his insightful work and his explanations given via email.

References

1. Iliano Cervesato and Frank Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.
2. Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 259–273, Bonn, Germany, September 1996. MIT Press.
3. Dominic Duggan. Unification with extended patterns. *Theoretical Computer Science*, 206(1-2):1–50, 1998.
4. Conal Elliott. *Extensions and Applications of Higher-Order Unification*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as Technical Report CMU-CS-90-134.
5. Roland Fettig and Bernd Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In *7th International Conference on Rewriting Techniques and Applications (RTA'96), New Brunswick, NJ, USA*, volume 1103 of *Lecture Notes in Computer Science (LNCS)*, pages 347–361. Springer, 1996.
6. Healfdene Goguen. Justifying algorithms for $\beta\eta$ conversion. In *Proc. of the 8th Int. Conf. on Foundations of Software Science and Computational Structures, FoSSaCS 2005*, volume 3441 of *Lect. Notes in Comput. Sci.*, pages 410–424. Springer, 2005.
7. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981.
8. Dale Miller. Unification of simply typed lambda-terms as logic programming. In *Eighth International Logic Programming Conference*, pages 255–269, Paris, France, June 1991. MIT Press.

9. Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.
10. Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Dept of Comput. Sci. and Engrg., Chalmers, Göteborg, Sweden, September 2007.
11. Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, September 2007. Technical Report 33D.
12. Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 202–206. Springer, 1999.
13. Brigitte Pientka. *Tabled higher-order logic programming*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 2003. CMU-CS-03-185.
14. Brigitte Pientka and Joshua Dunfield. Beluga: a framework for programming and reasoning with deductive systems (System Description). In *5th International Joint Conference on Automated Reasoning (IJCAR'10)*, Lecture Notes in Artificial Intelligence (LNAI), 2010.
15. Adam Poswolsky and Carsten Schürmann. System description: Delphin—a functional programming language for deductive systems. In *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'08)*, volume 228 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, pages 135–141. Elsevier, 2009.
16. Jason Reed. Higher-order constraint simplification in dependent type theory. In *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'09)*, 2009.
17. Jason Reed. *A Hybrid Logical Framework*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2009.
18. Anders Schack-Nielson and Carsten Schürmann. Pattern unification for the lambda calculus with linear and affine types. In *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'10)*, volume 34 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 101–116, July 2010.
19. Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgements and properties. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2003.

A Hereditary substitution

Normal forms are maintained through the use of hereditary substitution, written as $[N/x]_A(B)$ to guarantee that when we substitute term N which has type A for the variable x in the type B, we obtain a type B' which is in normal form. Hereditary substitutions continue to substitute, if a redex is created; for example, when replacing naively x by $\lambda y.c\, y$ in the object $x\, z$, we would obtain $(\lambda y.c\ y)\ z$ which is not in normal form and hence not a valid term in our grammar. Hereditary substitutions continue to substitute z for y in $c\, y$ to obtain $c\, z$ as a final result.

Hereditary substitution can be defined recursively considering the term to which the substitution operation is applied and the type of the object which is being substituted. We define the hereditary substitution operations for normal object, neutral objects and substitutions. The hereditary substitution operations will be defined by nested induction, first on the structure of the type A and second on the structure of the objects N, R, and σ. In other words, we either go to a smaller type, in which case the objects themselves can become larger, or the type remains the same and the objects become smaller. We write $A \leq B$ and $A < B$ if A occurs in B (as a proper sub-expression in the latter case)[4]. Hereditary substitution is defined in Figure 7. For an in depth discussion, we refer the reader to Nanevski et al. [9].

If the original term is not well-typed, a hereditary substitution, though terminating, cannot always return a meaningful term. We formalize this as failure to return a result. However, on well-typed terms, hereditary substitution will always return well-typed terms. The definition for single hereditary substitutions can be easily extended to simultaneous substitutions substitution written as $[\sigma]_\Psi(M)$. We annotate the substitution with the sub-script Ψ for two reasons. First, σ itself does not carry its domain and hence we will look up the instantiation for a variable x in σ/Ψ. Second, we rely on the type of x in the context Ψ to guarantee that applying σ to an object terminates. Either we apply σ to sub-expressions or the type of the object we substitute will be smaller. Subsequently, we often omit the typing subscript at the substitution operation for better readability.

[4] To ensure termination, it suffices to rely on type approximations of the dependent type; we leave this out from the discussion.

Normal Terms / Types

$[M/x]_A(\Pi y{:}B_1.B_2)$	$= \Pi y{:}B'_1.B'_2$	where $B'_1 = [M/x]_A(B_1)$ and $B'_2 = [M/x]_A(B_2)$, $y \notin \mathsf{FV}(M)$, and $y \neq x$
$[M/x]_A(\Sigma y{:}B_1.B_2)$	$= \Sigma y{:}B'_1.B'_2$	where $B'_1 = [M/x]_A(B_1)$ and $B'_2 = [M/x]_A(B_2)$, $y \notin \mathsf{FV}(M)$, and $y \neq x$
$[M/x]_A(\mathsf{type})$	$= \mathsf{type}$	
$[M/x]_A(\lambda y.N)$	$= \lambda y.N'$	where $[M/x]_A(N) = N'$, $y \notin \mathsf{FV}(M)$, and $y \neq x$
$[M/x]_A(N_1, N_2)$	$= (N'_1, N'_2)$	where $[M/x]_A(N_1) = N'_1$ and $[M/x]_A(N_1) = N'_1$
$[M/x]_A(R)$	$= M'$	if $[M/x]_A(R) = M' : A'$
$[M/x]_A(R)$	$= R'$	if $[M/x]_A(R) = R'$
$[M/x]_A(N)$	fails	otherwise

Neutral terms

$[M/x]_A(x)$	$= M : A$	
$[M/x]_A(y)$	$= y$	if $y \neq x$
$[M/x]_A(u[\sigma])$	$= u[\sigma']$	where $[M/x]_A(\sigma) = \sigma'$
$[M/x]_A(R\,N)$	$= R'\,N'$	where $[M/x]_A(R) = R'$ and $[M/x]_A(N) = N'$
$[M/x]_A(R\,N)$	$= M'' : B$	if $[M/x]_A(R) = \lambda y.M' : \Pi y{:}A_1.B$ where $\Pi x{:}A_1.B \leq A$ and $[M/x]_A(N) = N'$ and $[N'/y]_{A_1}(M') = M''$
$[M/x]_A(\pi R)$	$= \pi R'$	where $[M/x]_A(R) = R'$
$[M/x]_A(\mathsf{fst}\,R)$	$= M_1 : B_1$	where $[M/x]_A(R) = (M_1, M_2) : \Sigma x{:}B_1.B_2$ where $\Sigma x{:}B_1.B_2 \leq A$
$[M/x]_A(\mathsf{snd}\,R)$	$= M_2 : B_2$	where $[M/x]_A(R) = (M_1, M_2) : \Sigma x{:}B_1.B_2$ where $\Sigma x{:}B_1.B_2 \leq A$
$[M/x]_A(R)$	fails	otherwise

Substitution

$[M/x]_A(\cdot)$	$= \cdot$	
$[M/x]_A(\sigma\,,\,N)$	$= (\sigma'\,,\,N')$	where $[M/x]_A(\sigma) = \sigma'$ and $[M/x]_A(N) = N'$
$[M/x]_A(\sigma)$	fails	otherwise

Fig. 7. Hereditary substitutions for LF objects with contextual variables