Under consideration for publication in Math. Struct. in Comp. Science

Extensions to Miller's Pattern Unification for Dependent Types and Records

Andreas ABEL and Brigitte PIENTKA
Department of Computer Science and Engineering, Gothenburg University
School of Computer Science, McGill University, Montreal, Canada

Received 2 May 2018

Several extensions to Miller's algorithm for higher-order pattern unification are presented. The main extension is to dependent record types which are decomposed into strong Sigma-types and an extensional unit type. Our algorithm takes type isomorphisms into account to translate unification problems containing dependent record types into those involving only dependent function types. Further extensions such as postponement of constraints, handling of non-linear patterns, and pruning improve the performance of the unifier in practical type reconstruction problems.
Our solution takes the form of a inference system that applies small refinements to a constraint set until an inconsistency has surfaced or a (partial) solution been reached. The article describes the individual refinement steps of our algorithm using numerous examples together with the correctness proof of the algorithm.

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 [Goldfarb, 1981], Miller [1991] 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, the pure pattern fragment is too restrictive for many applications. Systems such as Abella [Gacek, 2008], Agda [2018], Beluga [Pientka and Dunfield, 2010, Pientka and Cave, 2015], λProlog [Nadathur and Mitchell, 1999, Nadathur and Linnell, 2005], and Twelf [Pfenning and Schürmann, 1999] solve eagerly these sub-problems which fall into the pattern fragment and delay sub-problems outside the pattern fragment until more information has been gathered. The additional information might simplify the delayed constraints such that they also fall into the pattern fragment; we speak of the *dynamic* pattern fragment when postponement is involved [Michaylov and Pfenning, 1992]. As a consequence, the unification algorithm is best described as an *inference system* [Shankar, 2005]. Initial unification constraints are refined in small steps ("inferences") until a (partial) solution is found or an inconsistency has surfaced. Alternatively, we could say we apply non-deterministically *rewrite rules* to the set of constraints until no further progress is possible.

In addition, we need higher-order unification beyond the pure λ^{Π}-calculus. In Beluga and Twelf, Σ-types are used to group assumptions together. Agda supports Σ-types in form of records with associated η-equality in its general form. To handle also Σ-types, extra ingredients are needed in the unification algorithm. For instance, following terms may be seen as equivalent:

1 $\lambda y_1.\lambda y_2.\, X\,(y_1, y_2)$,
2 $\lambda y.\, X\,(\mathsf{fst}\ y)\,(\mathsf{snd}\ y)$ and
3 $\lambda y_1.\lambda y_2.\, X\, y_1\, y_2$.

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 (1), we replace X with $\lambda y.\, X'\,(\mathsf{fst}\ y)\,(\mathsf{snd}\ y)$; for term (2), we unfold y which stands for a pair and replace y with (y_1, y_2).

In this article, 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. [2008] and the second author [Pientka, 2003], our description takes advantage of modelling meta-variables as closures; instead of directly considering a meta-variable X at function type $\Pi\vec{x}{:}\vec{A}.B$ which is applied to $\vec{x}$, we describe them as contextual objects, i.e., objects of type B in a context $\vec{x}{:}\vec{A}$, which are associated with a delayed substitution for the local context $\vec{x}{:}\vec{A}$.[†] This allows us to give a high-level description and analysis following Dowek et al. [1996], 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 work also avoids some of the other shortcomings; as pointed out by Reed [2009b], the algorithm sketched by Dowek et al. [1996] fails to terminate on some inputs. We give a clear specification of the pruning which eliminates bound variable dependencies for the dependently

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

typed case and show correctness of the unification algorithms in three steps: first, we show that it terminates, then, we show that the transformations in our unification algorithm preserve types, and, finally, that each transition neither destroys nor creates (additional) solutions.

This is the extended version of a published conference paper [Abel and Pientka, 2011].

2. $\lambda^{\Pi\Sigma}$-calculus with meta-variables

In this paper, we are considering an extension of the $\lambda^{\Pi\Sigma}$-calculus [Pfenning, 1989] with meta-variables. Its grammar is given in Fig. 1. Besides atomic types P of the form $\mathsf{a}\, M_1 \dots M_n$ which can be introduced by constructors c, it features dependent function types $\Pi x{:}A.\, B$ with abstraction $\lambda x.\, M$ and application $R\, N$ and dependent pair types $\Sigma x{:}A.\, B$ with tupling $(M,\ N)$ and projection $\pi\, R$, where π can be the first (fst) or second (snd) projection. As $\lambda^{\Pi\Sigma}$ is strongly normalizing, it is sufficient to consider β-normal forms only.[‡] Consequently, only neutral terms R can be in elimination position, where neutrals are rigid heads H possibly eliminated by applications and/or projections.

Variables	x, y, z		
Meta-variables	u, v, w		
Sorts	s	$::=$	$\mathsf{type} \mid \mathsf{kind}$
Atomic types	P, Q	$::=$	$\mathsf{a}\, \vec{M}$
Types	A, B, C, D	$::=$	$P \mid \Pi x{:}A.B \mid \Sigma x{:}A.B$
Kinds	κ	$::=$	$\mathsf{type} \mid \Pi x{:}A.\kappa$
(Rigid) heads	H	$::=$	$\mathsf{a} \mid \mathsf{c} \mid x$
Projections	π	$::=$	$\mathsf{fst} \mid \mathsf{snd}$
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	σ, τ	$::=$	$\cdot \mid \sigma, M$
Variable substitutions	ρ, ξ	$::=$	$\cdot \mid \rho, x$
Contexts	Ψ, Φ, Γ	$::=$	$\cdot \mid \Psi, x{:}A$
Meta substitutions	θ, η	$::=$	$\cdot \mid \theta, \hat{\Psi}.M/u$
Meta contexts	Δ	$::=$	$\cdot \mid \Delta, u{:}A[\Psi]$

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

Meta-variables appear in terms as closures $u[\sigma]$ which consist of a meta-variable u under a suspended explicit substitution σ. The term $\lambda x\, y\, z.\, X\, x\, y$ with the meta-variable X of 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. This meta-variable u can be replaced by a *contextual object* such as $x, y.\, x\, (\mathsf{suc}\, y)$. In general,

[‡] In our terminology, β subsumes all computation steps; beyond application reduction $(\lambda x.\, M)\, N =_\beta [N/x]M$ also the projection reductions $\mathsf{fst}\,(M, N) =_\beta M$ and $\mathsf{snd}\,(M, N) =_\beta N$.

a contextual object $\boxed{\hat{\Psi}.\, M}$ is a term M whose free variables have to be contained in the variable list $\boxed{\hat{\Psi}}$. (In the example, $\hat{\Psi} = x, y$ and $M = x\,(\mathsf{suc}\,y)$.) The use of contextual objects instead of closed terms eliminates the need to craft a λ-prefix for the instantiation of meta-variables, avoiding the creation of administrative β-redexes. In general, meta-variable u of *contextual type* $\boxed{A[\Psi]}$ stands for a contextual object $\hat{\Psi}.M$ where $\hat{\Psi}$ is the domain of Ψ, i. e., the list of variables declared in Ψ in the correct order. The use of the prefix $\hat{\Psi}$ allows us to rename the free variables occurring in M if necessary.

A *signature* Σ is a collection of declarations, which take one of the forms: $\mathsf{a} : \kappa$ (type family declaration) or $\mathsf{c} : A$ (constructor declaration). While signatures fill $\lambda^{\Pi\Sigma}$ with "life", i. e., data types and structures, they do not matter much for our studies of unification in this article. The whole development is parametrized by a fixed signature Σ which we usually suppress in typing and other judgements.

Because *variable substitutions* ρ play a special role in the formulation of our unification algorithm, we recognize them as a subclass of general substitutions σ. Weakening substitutions $\boxed{\mathsf{wk}_\Phi}$, which are a special case of variable substitutions, are defined recursively by $\mathsf{wk}_{\cdot} = (\cdot)$ and $\mathsf{wk}_{\Phi,x:A} = (\mathsf{wk}_\Phi, x)$. The subscript Φ is dropped when unambiguous. Incidentially, $\mathsf{wk}_\Phi = \hat{\Phi}$, but conceptually, one is a substitution and one a list of binders, thus, we keep them separate. If and only if Φ is a sub-context of Ψ modulo η-equality, then wk_Φ is a well-formed substitution in Ψ, i.e., $\Psi \vdash \mathsf{wk}_\Phi : \Phi$ holds (see Fig. 2).

We write $\boxed{E[M]}$ 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 [Cervesato and Pfenning, 2003], simply writing $H\, M_1 \dots M_n$. Evaluation contexts are the proper generalization of spines to projections. [§]

Occurrences and free variables. If α, β are syntactic entities such as evaluation contexts, terms, or substitutions, $\alpha, \beta ::= E \mid R \mid M \mid \sigma$, we write $\boxed{\alpha\{\beta\}}$ if β is a part of α. If we subsequently write $\alpha\{\beta'\}$ then we mean to replace the indicated occurrence of β by β'. We say that 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 $\mathsf{c}\,(u[y_1])\,(x_1\,x_2)\,(\lambda z.\, z\, x_3\, v[y_2, w[y_3]])$ 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 [2009b] we indicate rigid occurrences by $\boxed{\alpha\{\beta\}^{\mathsf{rig}}}$ and strongly rigid occurrences by $\boxed{\alpha\{\beta\}^{\mathsf{srig}}}$.

Flexible variable occurrences can vanish by instantiation of meta-variables, rigid ones not. In our example, y_1 will disappear by the substitution $y_1.\mathsf{c}'/u$. Rigid variables might disappear by instantiation of other free variables and subsequent normalization. For instance, x_2 disappears when we substitute $\lambda x_2.\mathsf{c}'$ for x_1. However, strongly rigid ones will only disappear when they are instantiated themselves, not through other variable instantiations.

[§] See also Schack-Nielsen and Schürmann [2010] for a similar generalization in the linear setting.

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

Typing and equality. Fig. 2 lists the typing judgements and rules. Separating the typing rules for neutrals, whose type can be inferred ($\Rightarrow$ judgments), from the typing rules for normal expressions, which need the type as input ($\Leftarrow$ judgement), we obtain a simple bidirectional type checking algorithm which is complete for β-normal forms. In contrast to other bidirectional formulations of LF [Harper and Licata, 2007], our terms need not be η-long. The judgment $\boxed{A =_\eta C}$ (rules omitted) compares A and C modulo η, i.e., modulo $R = \lambda x.\, R\, x$ (when $x \notin \mathsf{FV}(R)$) and $R = (\mathsf{fst}\, R\,,\ \mathsf{snd}\, R)$.

Hereditary substitution and meta substitution. For α a well-typed entity in context Ψ and $\Delta; \Phi \vdash \sigma : \Psi$ a well-formed substitution, we define a simultaneous substitution operation $\boxed{[\sigma]_\Psi \alpha}$ 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* [Watkins et al., 2003], proceeds by resolving newly created redexes on the fly through further substitutions. Both Agda [2018] and Beluga [Pientka and Dunfield, 2010, Pientka and Cave, 2015] use that strategy, as well as other theoretical investigations of logical frameworks [Harper and Licata, 2007]. Single hereditary substitution $\boxed{[N/x]_A \alpha}$ 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 and simply write $\boxed{[\sigma]\alpha}$ and $\boxed{[N/x]\alpha}$, resp.

The *meta-substitution* operation, i. e., substitution of meta-variables by contextual objects, is written as $\boxed{[\![\hat{\Psi}.M/u]\!]N}$ and the simultaneous meta substitution as $\boxed{[\![\theta]\!]N}$. Both operations restore β-normality. In the particular case when we apply $\hat{\Psi}.M/u$ to $u[\sigma]$, we first substitute $\hat{\Psi}.M$ for u in σ to obtain σ'. Subsequently, we continue to apply σ' to M hereditarily to obtain M'. Note that meta substitutions are "closed" wrt. LF variables, i. e., $\mathsf{FV}(\theta) = \emptyset$. Thus, they go under binders unaltered, and we have $[\![\theta]\!](\lambda x N) = \lambda x.\, [\![\theta]\!]N$.

3. Constraint-based unification

In this section, we define our unification algorithm for $\lambda^{\Pi\Sigma}$ in the style of an inference system [Shankar, 2005], i. e., using rewrite rules which solve constraints incrementally. Constraints K and sets of constraints $\mathcal{K}$ are defined as follows:

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.
Constraint sets	$\mathcal{K} ::= K \mid \mathcal{K} \wedge K$	(modulo laws of conjunction).

Neutral terms/types $\boxed{\Delta;\Psi \vdash R \Rightarrow A}$ (Δ and Ψ fixed)

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

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

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

Normal terms $\boxed{\Delta;\Psi \vdash M \Leftarrow A}$ (Δ fixed)

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

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

Substitutions $\boxed{\Delta;\Psi \vdash \sigma \Leftarrow \Psi'}$ (Δ and Ψ fixed)

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

LF types and kinds $\boxed{\Delta;\Psi \vdash A \Leftarrow s}$ (Δ fixed)

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

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

LF typing contexts $\boxed{\Delta \vdash \Psi\ \mathsf{ctx}}$ (Δ fixed)

$$\frac{}{\Delta \vdash \cdot\ \mathsf{ctx}} \qquad \frac{\Delta \vdash \Psi\ \mathsf{ctx} \qquad \Delta;\Psi \vdash A \Leftarrow \mathsf{type}}{\Delta \vdash \Psi, x{:}A\ \mathsf{ctx}}$$

Meta substitutions $\boxed{\Delta \vdash \theta \Leftarrow \Delta'}$ (Δ fixed)

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

Meta contexts $\boxed{\vdash \Delta\ \mathsf{mctx}}$

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

Fig. 2. Typing rules for $\lambda^{\Pi\Sigma}$ with meta-variables

Our basic constraints are quadruples $\Psi \vdash M = N : C$. The type annotation $\Psi \vdash C$ serves two purposes: First, we need types to direct any hereditary substitutions we employ during constraint solving. Secondly, 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 dependencies in this presentation to scale this work from $\lambda^{\Pi\Sigma}$ to non-erasable dependent types such as featured by Agda.

Constraints of the form $\Psi \mid R{:}A \vdash E = E'$ specify an equality between evaluation contexts. These are intermediate constraints used for the decomposition of neutral terms. The meaning the latter constraint is $\Psi \vdash E[R] = E[R'] : C$ where C is the type of $E[R]$ which can be computed via $\Delta; \Psi \vdash E[R] \Rightarrow C$.

A unification problem is described by a pair $\boxed{\Delta \Vdash \mathcal{K}}$ where meta-variable context Δ contains at least the typings of the meta-variables occurring 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 constraint nor in any type in Δ (nor in its own solution M).

A set of constraints $\mathcal{K}$ is *well-formed*, $\boxed{\Delta \Vdash \mathcal{K}\ \mathsf{wf}}$, if each constraint $K \in \mathcal{K}$ is well-typed. However, requiring strictly well-typed constraints at every point through the algorithm would limit its capabilities considerably. Consider a meta-variable $u : [x{:}\mathsf{bool}]A$ and constraint

$$\Psi \vdash (u[x],\, M) = (u[\mathsf{true}],\, N) : \Sigma x{:}A.\, P\, x.$$

We would like to decompose it into the two constraints $\Psi \vdash u[x] = u[\mathsf{true}] : A$ and $\Psi \vdash M = N : P\, u[x]$. Well-typedness of the second constraint would require $P\, u[x] =_\eta P\, u[\mathsf{true}]$ which does not hold. However, it holds for every solution of u that satisfies the first constraint.

The ability to postpone constraints and revisit them later is crucial for practical type reconstruction in the presence of dependent types. Thus, we give up well-typedness and adopt *typing modulo constraints* in the formulation of Reed [2009b] for LF.

3.1. *Typing modulo*

For all typing judgments $\Delta; \Psi \vdash J$ defined previously, we define $\boxed{\Delta; \Psi \vdash_{\mathcal{K}} J}$ by the same rules as for $\Delta; \Psi \vdash J$ except replacing eta equality $=_\eta$ with equality modulo $=_{\mathcal{K}}$. We write $\boxed{\alpha =_{\mathcal{K}} \beta}$ if we have $[\![\theta]\!]\alpha =_\eta [\![\theta]\!]\beta$ for any ground solution θ of $\mathcal{K}$. To put it differently, if we can solve $\mathcal{K}$, we can establish that α is equal to β.

The following lemmas proven by Reed [2009b] 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. We first prove that typing modulo is preserved under equality modulo.

Lemma 3.1 (Conversion modulo). Let $\Delta_0 \Vdash \mathcal{K}$ and $\Delta =_{\mathcal{K}} \Delta'$ and $\Psi =_{\mathcal{K}} \Phi$ and $A =_{\mathcal{K}} B$.

1 If $\Delta; \Psi \vdash_{\mathcal{K}} M \Leftarrow A$ then $\Delta'; \Phi \vdash_{\mathcal{K}} M \Leftarrow B$.
2 If $\Delta; \Psi \vdash_{\mathcal{K}} R \Rightarrow A$ then $\Delta'; \Phi \vdash_{\mathcal{K}} R \Rightarrow B$.
3 If $\Delta; \Psi \vdash_{\mathcal{K}} \sigma \Leftarrow \Psi'$ and $\Psi' =_{\mathcal{K}} \Phi'$ then $\Delta'; \Phi \vdash_{\mathcal{K}} \sigma \Leftarrow \Phi'$.

Proof. We generalize the statement to types and kinds and prove them by simultaneous induction on the typing derivation. □

Lemma 3.2 (Substitution principle modulo). Let $\Delta \Vdash \mathcal{K}$. If $\Delta; \Psi \vdash_{\mathcal{K}} M \Leftarrow A$ and $\Delta; \Psi, x{:}B, \Psi' \vdash_{\mathcal{K}} J$ and $A =_{\mathcal{K}} B$ then $\Delta; \Psi, [M/x]_A \Psi' \vdash_{\mathcal{K}} [M/x]_A J$.

Proof. This proof follows essentially the proof by Nanveski et al. [2008] and we generalize the property to types, kinds, and contexts. Since we prove the substitution lemma modulo $A =_{\mathcal{K}} B$, we use Lemma 3.1 when we consider the variable case. □

Lemma 3.3 (Meta-substitution principle modulo). Let $\Delta_0 \Vdash \mathcal{K}$. If $\Delta_1 \vdash_{[\![\theta]\!]\mathcal{K}} \theta \Leftarrow \Delta_0$ and $\Delta_0; \Phi \vdash_{\mathcal{K}} J$ then $\Delta_1; [\![\theta]\!]\Phi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]J$.

Proof.
The proof proceeds by induction on $\Delta_0; \Phi \vdash_{\mathcal{K}} J$. All cases are by inversion, appeal to the induction hypothesis (i.h.), reassembling the result and if necessary using Lemma 3.1 (see the case for meta-variables). We detail two cases:

Case Transition between inference and checking.

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

$\Delta_1; [\![\theta]\!]\Phi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]R \Rightarrow [\![\theta]\!]C_1$ by i.h.
$[\![\theta]\!]C_1 =_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]C_2$ by i.h. $C_1 =_{\mathcal{K}} C_2$
$\Delta_1; [\![\theta]\!]\Phi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]R \Leftarrow [\![\theta]\!]C_2$.

Case Meta-variable.

$$\mathcal{D} = \frac{\Delta; \Psi \vdash_{\mathcal{K}} \sigma \Leftarrow \Psi' \quad u{:}B[\Psi'] \in \Delta}{\Delta; \Psi \vdash_{\mathcal{K}} u[\sigma] \Leftarrow [\sigma]_{\Psi'}(B)}$$

For $\hat{\Psi}'.M/u \in \theta$ and $u{:}B[\Psi'] \in \Delta$ we have $\Delta_1; [\![\theta]\!]\Psi' \vdash_{\mathcal{K}} M \Leftarrow [\![\theta]\!]B$.
$\Delta_1; [\![\theta]\!]\Psi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]\sigma \Leftarrow [\![\theta]\!]\Psi'$ by i.h.
$[\![\theta]\!](u[\sigma]) = [[\![\theta]\!]\sigma]_{\Psi'}(M)$ by definition
$\Delta_1; [\![\theta]\!]\Psi \vdash_{[\![\theta]\!]\mathcal{K}} [[\![\theta]\!]\sigma]M \Leftarrow [[\![\theta]\!]\sigma]([\![\theta]\!]B)$ by ord. subst. lemma
$\Delta_1; [\![\theta]\!]\Psi \vdash_{[\![\theta]\!]\mathcal{K}} [[\![\theta]\!]\sigma]M \Leftarrow [\![\theta]\!]([\sigma]B)$ by definition of meta substitution □

A unification problem $\boxed{\Delta \Vdash \mathcal{K} \text{ is well-formed}}$ if $\Delta\ \mathsf{mctx}_{\mathcal{K}}$ and all constraints $(\Psi \vdash M = N : C) \in \mathcal{K}$ are well-typed modulo $\mathcal{K}$, i.e., $\Delta \vdash_{\mathcal{K}} \Psi\ \mathsf{ctx}$ and $\Delta; \Psi \vdash_{\mathcal{K}} M \Leftarrow C$ and $\Delta; \Psi \vdash_{\mathcal{K}} N \Leftarrow C$ and $\Delta; \Psi \vdash_{\mathcal{K}} C \Leftarrow \mathsf{type}$. 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*

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* (Fig. 3) apply to a single constraint, decomposing it and molding it towards a pattern by η-contraction and projection elimination. Decomposition of neutral terms is defined using evaluation contexts to have direct access to the head.

Decomposition of functions

$$\begin{array}{lll}
\Psi \vdash \lambda x.M = \lambda x.N : \Pi x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi, x{:}A \vdash M = N : B \\
\Psi \vdash \lambda x.M = R : \Pi x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi, x{:}A \vdash M = R\,x : B \\
\Psi \vdash R = \lambda x.M : \Pi x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi, x{:}A \vdash R\,x = M : B
\end{array}$$

Decomposition of pairs

$$\begin{array}{lll}
\Psi \vdash (M_1\,, M_2) = (N_1\,, N_2) : \Sigma x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi \vdash M_1 = N_1 : A \wedge \Psi \vdash M_2 = N_2 : [M_1/x]B \\
\Psi \vdash (M_1\,, M_2) = R : \Sigma x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi \vdash M_1 = \mathsf{fst}\,R : A \wedge \Psi \vdash M_2 = \mathsf{snd}\,R : [M_1/x]B \\
\Psi \vdash R = (M_1\,, M_2) : \Sigma x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi \vdash \mathsf{fst}\,R = M_1 : A \wedge \Psi \vdash \mathsf{snd}\,R = M_2 : [M_1/x]B
\end{array}$$

Decomposition of neutrals

$$\begin{array}{lll}
\Psi \vdash E[H] = E'[H] : C & \mapsto_{\mathsf{d}} & \Psi \mid H{:}A \vdash E = E' \text{ where } \Psi \vdash H \Rightarrow A \\
\Psi \vdash E[H] = E'[H'] : C & \mapsto_{\mathsf{d}} & \bot \text{ if } H \neq H'
\end{array}$$

Decomposition of evaluation contexts

$$\begin{array}{lll}
\Psi \mid R : A \vdash \bullet = \bullet & \mapsto_{\mathsf{d}} & \top \\
\Psi \mid R : \Pi x{:}A.\,B \vdash E[\bullet\, M] = E'[\bullet\, M'] & \mapsto_{\mathsf{d}} & \Psi \vdash M = M' : A \wedge \Psi \mid R\,M : [M/x]B \vdash E = E' \\
\Psi \mid R : \Sigma x{:}A.\,B \vdash E[\mathsf{fst}\,\bullet] = E'[\mathsf{fst}\,\bullet] & \mapsto_{\mathsf{d}} & \Psi \mid \mathsf{fst}\,R\ : A \vdash E = E' \\
\Psi \mid R : \Sigma x{:}A.\,B \vdash E[\mathsf{snd}\,\bullet] = E'[\mathsf{snd}\,\bullet] & \mapsto_{\mathsf{d}} & \Psi \mid \mathsf{snd}\,R : [\mathsf{fst}\,R/x]B \vdash E = E' \\
\Psi \mid R : \Sigma x{:}A.\,B \vdash E[\pi\,\bullet] = E'[\pi'\,\bullet] & \mapsto_{\mathsf{d}} & \bot \text{ if } \pi \neq \pi'
\end{array}$$

Orientation

$$\begin{array}{lll}
\Psi \vdash M = u[\sigma] : C \text{ with } M \neq v[\ldots] & \mapsto_{\mathsf{d}} & \Psi \vdash u[\sigma] = M : C
\end{array}$$

η-Contraction

$$\begin{array}{lll}
\Psi \vdash u[\sigma\{\lambda x.\,R\,x\}] = N : C & \mapsto_{\mathsf{e}} & \Psi \vdash u[\sigma\{R\}] = N : C \\
\Psi \vdash u[\sigma\{(\mathsf{fst}\,R, \mathsf{snd}\,R)\}] = N : C & \mapsto_{\mathsf{e}} & \Psi \vdash u[\sigma\{R\}] = N : C
\end{array}$$

Eliminating projections

$$\begin{array}{lll}
\Psi_1, x : \Pi\vec{y}{:}\vec{A}.\,\Sigma z{:}B.C,\ \Psi_2 & & \Psi_1, x_1 : \Pi\vec{y}{:}\vec{A}.\,B,\ x_2 : \Pi\vec{y}{:}\vec{A}.\,[(x_1\,\vec{y})/z]C,\ \Psi_2 \\
\quad \vdash u[\sigma\{\pi\,(x\,\vec{M})\}] = N : D & \mapsto_{\mathsf{p}} & \quad \vdash u[[\tau]\sigma] = [\tau]N : [\tau]D \\
\text{where } \pi \in \{\mathsf{fst}, \mathsf{snd}\} & & \text{where } \tau = [\lambda\vec{y}.\,(x_1\,\vec{y},\, x_2\,\vec{y})/x]
\end{array}$$

Fig. 3. Local simplification $\boxed{K \mapsto_m \mathcal{K}}$.

Decomposition of pairs could maybe more concisely defined by

$$\begin{array}{lll}
\Psi \vdash M = N : \Sigma x{:}A.\,B & \mapsto_{\mathsf{d}} & \Psi \vdash \mathsf{fst}@M = \mathsf{fst}@N : A \\
& \wedge & \Psi \vdash \mathsf{snd}@M = \mathsf{snd}@N : [\mathsf{fst}@M/x]B
\end{array}$$

where $\pi@M$ computes the β-normal form of $\pi\,M$. However, this would also apply to a constraint $\Psi \vdash R = R' : \Sigma x{:}A.\,B$ of neutral terms, just duplicating the work of decomposing the neutrals later. Similar reasoning justifies of choice of function decomposition rules.

The other *unification steps* (Fig. 4) work on a meta-variable and try to find an instantiation for it. We write $\boxed{\Delta \Vdash \mathcal{K}\ +\ \Phi \vdash u \leftarrow M : A}$ for instantiating the meta-variable u

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)

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

Lowering

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

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

Flattening Σ-types

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

Pruning

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

Same meta-variable

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

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

Fig. 4. Unification steps $\boxed{\Delta \Vdash \mathcal{K} \mapsto \Delta' \Vdash \mathcal{K}'}$.

with the term M both in the meta-context Δ and in the constraints $\mathcal{K}$. This abbreviation is defined in Fig. 4. 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.

To motivate our rules, let us consider some problems $\Psi \vdash u[\sigma] = M : C$ that fall outside 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 lhs to $u[y]$.

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

Lowering: $\Phi \vdash \mathsf{fst}\,(u[y]) = \mathsf{fst}\,y$ where $\Phi = (y : \Sigma x{:}A.\,B)$ and $u : (\Sigma x{:}A.\,B)[\Phi]$.
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, $u_1 : A[\Phi]$ and $u_2 : ([u_1[y]/x]B)[\Phi]$, yielding $\Phi \vdash u_1[y] = \mathsf{fst}\,y$.

Flattening Σ-types: $\Psi \vdash u[\lambda x.\,(z_1\,x\,,\,z_2\,x)] = \mathsf{g}\,z_1\,z_2$ where $\Psi(z_1) = \Pi x{:}A.\,B$ and $\Psi(z_2) = \Pi x{:}A.\,[z_1\,x/y]C$ and $u : P[z : \Pi x{:}A.\,\Sigma y{:}B.\,C]$.
By splitting z into two functions z_1, z_2 in the context of u and replacing u by $v : P[z_1 : \Pi x{:}A.\,B,\ z_2 : \Pi x{:}A.\,[z_1\,x/y]C]$ via meta sustitution $z.v[\lambda x.\,\mathsf{fst}\,z\,x,\ \lambda x.\,\mathsf{snd}\,z\,x]/u$, we arrive at $\Psi \vdash v[\lambda x.\,z_1\,x,\ \lambda x.\,z_2\,x] = \mathsf{g}\,z_1\,z_2$ 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 lhs can be ignored since x is not free on the rhs. We can solve this constraint by $u[x,y,z] = \mathsf{suc}\,z$.
However, in case of non-linearity we need to make sure that the solution is well-typed. Consider $u : (P\,z)[\Phi]$ where $\Phi = (x{:}A,\ y : P\,x,\ z{:}A)$ and constraint

$$x{:}A,\ y : P\,x \vdash u[x,y,x] = y : P\,x.$$

The solution $x{:}A,\ y : P\,x \vdash u \leftarrow y : P\,z$ is ill-typed. While the non-linear variables do not appear in the term of the rhs, they appear in the type and make the constraint well-typed.
In our type theory, the above constraint seems unsolvable, although we do not attempt a formal proof here. In richer type theories such as Agda one could imagine a solution for u which applies a type cast to y. Thus, we just leave such a constraint alone, instead of flagging unsolvability.

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 nested meta-variables. 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 [2009b] 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: $u[x,x] = \mathsf{suc}\ v[x]$ and $u'[x,x] = \mathsf{suc}\ v'[x,y]$.

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 *strongly* rigid occurrence in its own definition. This problem has only an infinite solution, independent of the variable substitutions u is applied to. To see this, consider the constraint $u[z, z, z] = \mathsf{suc}\, u[z, z, z]$ which is a special case of the original constraint with $[z/x, z/y]$. The only solution would be the infinite term $\mathsf{suc}(\mathsf{suc}(\dots))$. Since we require solutions to be finite, the occurs check signals unsolvability. Note that our occurs check is slightly more powerful than Reed's [2009b], since he only refutes such recursive occurrences when the left hand side is linear, e.g., of the form $u[x, y, z]$ with distinct variables x, y, z.

Reed [2009a, p. 105f] motivates why only *strongly* 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 $x, y \vdash u \leftarrow x$ and $x, y \vdash u \leftarrow y$. Other examples, like $u[x, x, z] = z$ which has unique solution $x, y, z \vdash u \leftarrow z$, suggest that we can ignore non-linear variable occurrences as long as they do not occur on the rhs. 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 or substitution $\alpha ::= M \mid R \mid \tau$ in context Ψ, we define the partial operation $[\rho/\hat{\Phi}]^{-1}\alpha$ by

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

and homeomorphic in all other cases by

$$\begin{array}{lcll}
[\rho/\hat{\Phi}]^{-1}(R\,M) & = & R'\,M' & \text{where } R' = [\rho/\hat{\Phi}]^{-1}R \text{ and } M' = [\rho/\hat{\Phi}]^{-1}M \\
 & & & \\
[\rho/\hat{\Phi}]^{-1}(\pi\,R) & = & \pi\,R' & \text{where } R' = [\rho/\hat{\Phi}]^{-1}R \\
[\rho/\hat{\Phi}]^{-1}(\lambda x.\,M) & = & \lambda x.\,M' & \text{if } x \notin \rho, \hat{\Phi} \text{ and } M' = [\rho, x \,/\, \hat{\Phi}, x]^{-1}M \\
[\rho/\hat{\Phi}]^{-1}(M\,,\,N) & = & (M'\,,\,N') & \text{where } M' = [\rho/\hat{\Phi}]^{-1}M \text{ and } N' = [\rho/\hat{\Phi}]^{-1}N \\
 & & & \\
[\rho/\hat{\Phi}]^{-1}(\cdot) & = & \cdot & \\
[\rho/\hat{\Phi}]^{-1}(\tau, M) & = & \tau', M' & \text{if } \tau' = [\rho/\hat{\Phi}]^{-1}\tau \text{ and } M' = [\rho/\hat{\Phi}]^{-1}M\ .
\end{array}$$

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

Lemma 3.4 (Inverse and meta-substitution commute).
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 3.5 (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 3.6 (Completeness of inverse substitution).
If $[\rho]_\Phi\alpha = \alpha'$ and for all $x, z \in \mathsf{FV}(\alpha)$ we have $y/x \in \rho$ but no other $y/z \in \rho$ then $\alpha = [\rho/\hat{\Phi}]^{-1}\alpha'$ exists.

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

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 ($\mathsf{FV}(\theta) = \emptyset$), we have $\mathsf{FV}(\sigma) \supseteq \mathsf{FV}([\![\theta]\!]\sigma) \supseteq \mathsf{FV}([[\![\theta]\!]\sigma]\theta(u)) \supseteq \mathsf{FV}([\![\theta]\!]M)$ (note that a hereditary substitution can remove free variables). 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 may fail for one of the following reasons:

1 Offending variables occur rigidly, like the y in $u[x] = \mathsf{c}\, y\, v[x,y]$. This constraint is unsolvable.

2 The flexible occurrence of an offending variable is under another meta-variable, like y 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.

3 The offending flexible occurrence could be eliminated by the correct solution. For instance, consider the case $u : C[x{:}A]$ and $v : C[z : (A \to A \to A) \to A]$ and constraint

$$x{:}A,\ y{:}A \vdash u[x] = v[\lambda k.\, k\, x\, y] : C.$$

The offending variable y occurs flexibly in this constraint and rigidly in v's substitution. If we pruned away v's dependency on z, we would lose the partial solution $\theta(v) = z.\, z\,(\lambda x \lambda y. x)$ which would simplify the constaint to $u[x] = x$. The point here is that y, although it occurs rigidly in $\lambda k.\, k\, x\, y$, is in an eliminateable position since the meta-substitution for v could place that term in a context that reduces y away. There are rigid occurences that cannot be eliminated in such a way, we call these occurrences *bad*, see judgement $\boxed{\mathsf{bad\_occ}_y\, N}$ in Fig. 5.

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 conceivable, but we have no data indicating that they strengthen unification significantly in practice. We employ the following judgments to define pruning (see Fig. 5):

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

where ρ is a variable substitution with domain Ψ and τ is a substitution from Ψ_1 to a context Ψ.

The second judgement is applied to subterms $v[\tau]$ of M to prune substitution τ with, say, domain Ψ_1. We look at each term N in τ which substitutes for an $x{:}A$ of Ψ_1. If N has a bad occurrence of a variable $y \notin \rho$, we discard the entry $x{:}A$ from the domain Ψ_1, thus, effectively removing N from τ. If N has no occurrence of such a 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. Pruning fails if N has an occurrence of a variable $y \notin \rho$ which is not bad, for instance, a

$\boxed{\mathsf{bad\_occ}_x\ M}$ Term M has a *non-eliminable* occurrence of variable x

$$\frac{}{\mathsf{bad\_occ}_x\ E[x]} \qquad \frac{\mathsf{bad\_occ}_x\ M}{\mathsf{bad\_occ}_x\ \lambda y.\, M}\ x \neq y \qquad \frac{\mathsf{bad\_occ}_x\ M_1 \qquad \mathsf{bad\_occ}_x\ M_2}{\mathsf{bad\_occ}_x\ (M_1, M_2)}$$

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

$$\frac{}{\mathsf{prune\_ctx}_\rho(\cdot\ /\ \cdot) \Rightarrow \cdot} \qquad \frac{\mathsf{prune\_ctx}_\rho(\tau\ /\ \Psi_1) \Rightarrow \Psi_2 \qquad \mathsf{bad\_occ}_x\ M \text{ for some } x \in \rho}{\mathsf{prune\_ctx}_\rho(\tau, M\ /\ \Psi_1, y{:}A) \Rightarrow \Psi_2}$$

$$\frac{\mathsf{prune\_ctx}_\rho(\tau\ /\ \Psi_1) \Rightarrow \Psi_2 \qquad \mathsf{FV}(M) \subseteq \rho \qquad A' = [\mathsf{wk}_{\Psi_2}/\hat{\Psi}_2]^{-1}A \text{ exists}}{\mathsf{prune\_ctx}_\rho(\tau, M\ /\ \Psi_1, x{:}A) \Rightarrow \Psi_2, x{:}A'}$$

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

$$\frac{v{:}B[\Psi_1] \in \Delta \quad \mathsf{prune\_ctx}_\rho(\tau\ /\ \Psi_1) \Rightarrow \Psi_2 \quad \Psi_2 \neq \Psi_1 \quad B' = [\mathsf{wk}_{\Psi_2}/\hat{\Psi}_2]^{-1}B \quad \eta = \hat{\Psi}_1.v'[\mathsf{wk}_{\Psi_2}]/v}{\Delta \vdash \mathsf{prune}_\rho(v[\tau]) \Rightarrow [\![\eta]\!](\Delta, v'{:}B'[\Psi_2]);\ \eta}$$

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

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

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

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

Fig. 5. Pruning.

flexible or rigid but eliminable occurrence. Examples:

- $\mathsf{prune\_ctx}_x(\mathsf{c}\,x,\ \ y\ \ /\ x'{:}A,\ y'{:}B) \ \Rightarrow\ x'{:}A$ bad occurrence y
- $\mathsf{prune\_ctx}_y(\mathsf{c}\,x,\ \ u[y]\ /\ x'{:}A,\ y'{:}B) \ \Rightarrow\ y'{:}B$ bad occurrence $\mathsf{c}\,x$
- $\mathsf{prune\_ctx}_y(\lambda z.\, z\,x,\ y\ \ /\ x'{:}A,\ y'{:}B)$ fails occ. of x eliminable
- $\mathsf{prune\_ctx}_y(u[x],\ \ y\ \ /\ x'{:}A,\ y'{:}B)$ fails flexible occurrence $u[x]$

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. The last check is strengthening B from Ψ_1 to Ψ_2 and can be implemented as $[\mathsf{wk}_{\Psi_2}/\hat{\Psi}_2]^{-1}B$ which exists whenever $\mathsf{FV}(B) \subseteq \hat{\Psi}_2$.

Lemma 3.7 (Bad occurrences stay). Let $\mathcal{D} :: \mathsf{bad\_occ}_x\, M$.

1 If $x \neq y$, then $\mathcal{D}' :: \mathsf{bad\_occ}_x\, [N/y]M$, and the derivation height of $\mathcal{D}'$ is the same as the one of $\mathcal{D}$.
2 $x \in \mathsf{FV}^{\mathsf{rig}}(E[M])$.
3 If $y \in \mathsf{FV}^{\mathsf{rig}}(N)$ then $x \in \mathsf{FV}^{\mathsf{rig}}([M/y]N)$.

Proof.

1 By induction on $\mathsf{bad\_occ}_x\, M$.
2 By induction on $\mathcal{D} :: \mathsf{bad\_occ}_x\, M$. In case $M = E'[x]$ we have trivially $x \in \mathsf{FV}^{\mathsf{rig}}(E[E'[x]])$. If $M = \lambda y.\, M'$ then either E is empty (trivial) or $E = E'[\bullet\, N]$. Then $E[M] = E'[[N/y]M']$, and by part 1 we get $\mathsf{bad\_occ}_x\, [N/y]M'$ with a smaller derivation height than $\mathcal{D}$. We conclude by induction hypothesis. If $M = (M_1, M_2)$ then either E is empty (trivial) or $E = E'[\mathsf{fst}\,\bullet]$ or $E = E'[\mathsf{snd}\,\bullet]$. In case of fst, $E[M] = E'[M_1]$ and we conclude by induction hypothesis. Case snd analogously.
3 By induction on β-normal form N. If N is a function $\lambda z.\, N'$ or a pair (N', N''), proceed with N'. If $N = E[y]$ then apply part 2. Otherwise N is a neutral which does not have y as head variable, which implies that hereditarily substituting y will yield a neutral with the same spine form. If $N = \pi\, R$ then $y \in \mathsf{FV}^{\mathsf{rig}}(R)$ and $[M/y]N = \pi\, R'$ with $R' = [M/y]R$. By induction hypothesis, $x \in \mathsf{FV}^{\mathsf{rig}}(R') = \mathsf{FV}^{\mathsf{rig}}([M/y]N)$. If N is an application $R'\, N'$ then $[M/y]N = R''\, N''$ with $R'' = [M/y]R'$ and $N'' = [M/y]N'$. Either $y \in \mathsf{FV}^{\mathsf{rig}}(R')$ or $y \in \mathsf{FV}^{\mathsf{rig}}(N')$ and we can conclude with the respective induction hypothesis, since $\mathsf{FV}^{\mathsf{rig}}(R''\, N'') = \mathsf{FV}^{\mathsf{rig}}(R'') \cup \mathsf{FV}^{\mathsf{rig}}(N'')$. □

Lemma 3.8 (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 $\mathsf{FV}([\tau]\mathsf{wk}_{\Psi_2}) \subseteq \rho$. 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 $\eta = \hat{\Psi}_1.v'[\mathsf{wk}_{\Psi_2}]/v$. Let $N = \theta(v)$. We have $\mathsf{FV}([[\![\theta]\!]\tau]N) \subseteq \rho$. 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 $\mathsf{FV}([[\![\theta]\!]\tau]x) \not\subseteq \rho$. This is in contradiction to $\mathsf{FV}([[\![\theta]\!]\tau]N) \subseteq \rho$. □

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 $[\rho]\mathsf{wk}_{\Phi'}$, which identical to $[\xi]\mathsf{wk}_{\Phi'}$, 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 3.9 (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 $\Delta; \Phi \vdash_{\mathcal{K}} \mathsf{wk}_{\Phi'} \Leftarrow \Phi'$ and $z \in \mathsf{dom}(\Phi')$ iff $\rho(z) = \xi(z)$.

Proof. By structural induction on the first derivation.

We consider the interesting case, where we actually retain a declaration $x{:}A$ because the two substitutions map x to the same variable y.

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

The main challenge is to show that type A is well-formed even in the subcontext Φ' of Φ, which is the case if $\mathsf{FV}(A) \subseteq \mathsf{dom}(\Phi')$.

From the assumption $\Delta; \Psi \vdash_{\mathcal{K}} (\rho, y) \Leftarrow (\Phi, x{:}A)$ we get $\Delta; \Psi \vdash_{\mathcal{K}} \rho \Leftarrow \Phi$ and $\Psi(y) =_{\mathcal{K}} [\rho]A$ by inversion, and the same for ξ. This allows us to apply the induction hypothesis, which yields $\Delta \vdash_{\mathcal{K}} \Phi'\ \mathsf{ctx}$ and $\Delta; \Phi \vdash_{\mathcal{K}} \mathsf{wk}_{\Phi'} \Leftarrow \Phi'$ and $\mathsf{dom}(\Phi') = \{z \mid \rho(z) = \xi(z)\}$. Since $[\rho]A =_{\mathcal{K}} \Psi(y) =_{\mathcal{K}} [\xi]A$, we have for all $z \in \mathsf{FV}(A)$ that $\rho(z) =_{\mathcal{K}} \xi(z)$. Since this is an equation between variables, the constraints $\mathcal{K}$ on meta-variables do not matter, and we even have $\rho(z) = \xi(z)$. Yet this means that $z \in \mathsf{dom}(\Phi')$, thus, we obtain $\Delta; \Phi' \vdash_{\mathcal{K}} A \Leftarrow \mathsf{type}$ by strengthening. This entails $\Delta \vdash_{\mathcal{K}} (\Phi', x{:}A)\ \mathsf{ctx}$, hence, trivially, $\Delta; (\Phi, x{:}A) \vdash_{\mathcal{K}} \mathsf{wk}_{\Phi', x:A} \Leftarrow (\Phi', x{:}A)$. Finally, $\mathsf{dom}(\Phi', x{:}A) = \{z \mid \rho(z) = \xi(z)\}$ follows from this property for Φ' and the fact $\rho(x) = y = \xi(x)$. □

Let us now reconsider the rule "Same meta-variable".

$$\begin{array}{lll} \Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = u[\xi] : C & \mapsto & \Delta, v{:}A[\Phi'] \Vdash \mathcal{K} + \Phi \vdash u \leftarrow v[\mathsf{wk}_{\Phi'}] : A \\ \quad u{:}A[\Phi] \in \Delta & & \quad \text{if } \rho \cap \xi : \Phi \Rightarrow \Phi' \end{array}$$

We have just shown that the resulting Φ' of computing the intersection of ρ and ξ, is indeed well-formed. 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 $\mathcal{K}$.

Note that intersection cannot easily be extended beyond variable substitutions. For instance, consider the intersection between the following substitutions σ and τ:

$$\underbrace{(\lambda f.\, f \circ f,\ \mathsf{g},\ N)}_{\sigma} \cap \underbrace{(\lambda f.\, f,\ \mathsf{g} \circ \mathsf{g},\ N)}_{\tau}\ :\ (x : A^4,\ y : A^2,\ z : P\,(x\,y))\ \Rightarrow\ ?$$

where $N : P\,(\mathsf{g} \circ \mathsf{g})$ and $A^4 := (A \to A) \to A \to A$ and $A^2 := A \to A$, and $f \circ g := \lambda n.\, f\,(g\,n)$ abbreviates function composition. We cannot remove x and y from the context only because the two substitutions differ at these positions. The resulting context $(z : P\,(x\,y))$ would be ill-typed. The difference to the variable case is that substitution is no longer injective, i.e., we can have $[\sigma]M = [\tau]M$ but $\sigma(x) \neq \tau(x)$ for some $x \in \mathsf{FV}(M)$.

4. Correctness

Theorem 4.1 (Termination). The algorithm terminates and results in one of the following states:

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

Proof. Let 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 [Goguen, 2005]. 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*.

Let 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, let the weight of constraint $\Phi \mid R{:}A \vdash E = E'$ be $(|E| + |E'|)\omega + |\Psi|$. Finally, let 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. $\square$

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 $\mathcal{K}$.

Before we prove that transitions preserve solutions, we first prove that there always exists a meta-substitution relating the original meta-variable context Δ_0 to the meta-variable context Δ_1 we transition to. It is useful to state this property in isolation, although it is also folded into Theorem 4.3.

Lemma 4.2. If $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ then there exists a meta-substitution θ s.t. $\Delta_1 \vdash_{\mathcal{K}_1} \theta \Leftarrow \Delta_0$.

Proof. By case analysis on the unification steps. □

We also 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 4.3 (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. $\Delta_1 \vdash_{\mathcal{K}_1} \theta' \Leftarrow \Delta_0$ 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 $[\![\theta_1]\!]\mathsf{wk}_{\Delta_0} \in \mathsf{Sol}(\Delta_0 \Vdash \mathcal{K}_0)$.

Proof. Proof by case analysis on the transitions. We only show the cases to prove that we are forward closed (statement 1). The second statement, that unification steps are backwards closed, is obvious: by if we transition from $\Delta_0 \Vdash \mathcal{K}_0$ to $\Delta_1 \Vdash \mathcal{K}_1$ then there exists a meta-substitution $\Delta_1 \vdash_{\mathcal{K}_1} \theta \Leftarrow \Delta_0$; hence, by composition $[\![\theta_1]\!]\theta$ is ground and is a solution for $\Delta_0 \Vdash \mathcal{K}_0$.

In all the following cases, θ_0 is a solution for $\mathcal{K}_0$.

- 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.
- Lowering. Let $u : (\Pi x{:}A.B)[\Phi] \in \Delta$ active and

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

Let $\hat{\Phi}.M := \theta_0(u)$ and observe $\cdot; [\![\theta_0]\!]\Phi \vdash M \Leftarrow [\![\theta_0]\!](\Pi x{:}A.B)$. By inversion on typing, $M = \lambda x.N$ and $\cdot; [\![\theta_0]\!](\Phi, x{:}A) \vdash N \Leftarrow [\![\theta_0]\!]B$. Hence, $(\hat{\Phi}, x).N$ is a solution for v. Choose $\theta' = \mathsf{wk}_{\Delta_0}$ and $\theta_1 = \theta_0, (\hat{\Phi}, x).N/v$. Since v is a new meta-variables, θ_1 is still a solution for the old state $\Delta \Vdash \mathcal{K}$.

The case for lowering Σ-types is similar.

- Flattening. Using the substitution lemma, solutions are preserved.
- 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. $[\![\theta']\!]\theta_p = \theta_0$.
- 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 $\cdot; [\![\theta_0]\!](\Phi) \vdash M \Leftarrow [\![\theta_0]\!]A$. 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 $\rho \cap \xi$; moreover, we must have $[\![\theta_0]\!]([\rho]A) = [\![\theta_0]\!]([\eta]A)$. Since meta-substitutions commute with variable substitutions, we have $[\rho]([\![\theta_0]\!]A) = [\eta]([\![\theta_0]\!]A)$, hence $\mathsf{FV}(M) = \hat{\Phi}_0$ and $\mathsf{FV}([\![\theta_0]\!]A) = \hat{\Phi}_0$ and $\cdot; [\![\theta_0]\!](\Phi_0) \vdash M \Leftarrow [\![\theta_0]\!]A$; choosing id_{Δ_0} for θ' and for $\theta_1 = \theta, \hat{\Phi}_0.M/v$ solutions are preserved.
- 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 4.4 (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 4.3, 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., $\Delta_0 \subseteq \Delta_1$ and that there always exists a meta-substitution θ which maps Δ_0 to Δ_1. Moreover, since transitions preserve solutions, if we have a solution for $\mathcal{K}_0$ there exists a solution for $\mathcal{K}_1$.

Lemma 4.5 (Transitions preserve typing). Let $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ and $\Delta_1 \vdash_{\mathcal{K}_1} \theta \Leftarrow \Delta_0$.

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

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 $\Delta_0 \Vdash_{\mathcal{K}_0} \mathcal{K}$ 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 $\Delta_0 \Vdash \mathcal{K}$ wf to mean $\Delta_0 \Vdash_{\mathcal{K}} \mathcal{K}$ wf.

Lemma 4.6 (Equations remain well-formed under meta-substitutions). If $\Delta_0 \Vdash \mathcal{K}$ wf and $\Delta_1 \vdash_{[\![\theta]\!]\mathcal{K}} \theta \Leftarrow \Delta_0$ then $\Delta_1 \Vdash [\![\theta]\!]\mathcal{K}$ wf.

Proof. By assumption $\Delta_0 \Vdash \mathcal{K}$ 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 3.3), we know $\Delta_1; [\![\theta]\!]\Psi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]M \Leftarrow [\![\theta]\!]A$ and $\Delta_1; [\![\theta]\!]\Psi \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]N \Leftarrow [\![\theta]\!]A$, and hence $\Delta_1 \vdash_{[\![\theta]\!]\mathcal{K}} [\![\theta]\!]K$ wf. $\square$

Lemma 4.7 (Well-formedness of equations is preserved by transitions). If $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ and $\Delta_0 \Vdash_{\mathcal{K}_0} \mathcal{K}$ wf then $\Delta_1 \vdash_{\mathcal{K}_1} \mathcal{K}$ wf.

Proof. By assumption $\Delta \Vdash_{\mathcal{K}_0} \mathcal{K}$ 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 4.5, 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 $\Delta \Vdash_{\mathcal{K}_1} \mathcal{K}$ wf. $\square$

Theorem 4.8 (Unification preserves well-formedness).
If $\Delta_0 \Vdash \mathcal{K}_0$ wf and $\Delta_0 \Vdash \mathcal{K}_0 \mapsto \Delta_1 \Vdash \mathcal{K}_1$ then $\Delta_1 \Vdash \mathcal{K}_1$ wf.

Proof. By case analysis on the transition rules and lemma 4.2.

– 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 $B_1 =_{\mathcal{K}_0} B_2$. In addition $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M \Leftarrow A$ and $\Delta_0; \Psi \vdash_{\mathcal{K}_0} M' \Leftarrow A$.
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 $\mathcal{K}_0$ and the fact that $\Psi \vdash M = M' : A$, 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 $\mathcal{K}_2 = \mathcal{K}_1 \wedge \Psi \mid R\; M : [M/x]B \vdash E = E'$.

– 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 3.8), we know that $\Delta_1 \vdash_{[\![\eta]\!]\mathcal{K}_0} \eta \Leftarrow \Delta_0$. By lemma 4.6, we know that $\Delta_1 \Vdash [\![\eta]\!](\mathcal{K}_0)$ wf.

– 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 3.9, 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 $(\Delta_0, v{:}A[\Phi_0]); \Phi \vdash_{\mathcal{K}_0} v[\mathsf{wk}_{\Phi_0}] \Leftarrow A$. Hence, $\Phi \vdash u \leftarrow v[\mathsf{wk}_{\Phi_0} : A$ is well-typed in Δ_0 modulo $\mathcal{K}_0$. Hence, $\theta = \hat{\Phi}.v[\mathsf{wk}_{\Phi_0}]/u$ is a well-formed meta-substitution. By lemma 3.3, we have $[\![\theta]\!]\Delta \Vdash ([\![\theta]\!]\mathcal{K}_0 \wedge [\![\theta]\!]\Phi \vdash u \leftarrow M : [\![\theta]\!]A)$ wf.

– 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 assumption 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 3.3, we have $[\![\theta]\!]\Delta_0 \Vdash [\![\theta]\!](\mathcal{K}_0 \wedge \Phi \vdash u \leftarrow M' : A)$ wf. □

5. Extension to unit type

To reduce dependently typed records to Σ-types, we still miss the empty record. In this section, we extend our unification algorithm to include the unit type $\mathbf{1}$. To account for the type-directed nature of equality with a unit type, we equip the η-equality judgement with context and type and write $\boxed{\Delta; \Phi \vdash M =_\eta N : A}$.

The challenge for our algorithm is that η-equality for the unit type does not preserve the set of free variables.

$$\frac{\Delta; \Phi \vdash M \Leftarrow \mathbf{1}}{\Delta; \Phi \vdash M =_\eta \star : \mathbf{1}}$$

Any term M of unit type is extensionally equal to the single inhabitant $\star : \mathbf{1}$. Thus, free variables may disappear by η-expansion. We need to disregard subterms of unit type during the occurrence check, otherwise we report unsolvability on actually solvable constraints.

Further, η-contraction needs to be type-directed to properly handle subterms of unit type. It is best formulated when we introduce the concept of *singleton type* which subsumes the unit type. For our purposes, a singleton type A sing is any type A that has exactly one inhabitant $\star^A$ up to η-equality.

$$\frac{}{\mathbf{1}\ \mathsf{sing}} \qquad \frac{B\ \mathsf{sing}}{\Pi x{:}A.\,B\ \mathsf{sing}} \qquad \frac{A\ \mathsf{sing} \quad B\ \mathsf{sing}}{\Sigma x{:}A.\,B\ \mathsf{sing}}$$

$$\star^{\mathbf{1}} = \star \qquad \star^{\Pi x:A.\,B} = \lambda x.\,\star^B \qquad \star^{\Sigma x:A.\,B} = (\star^A,\, \star^B)$$

Lemma 5.1 (Soundness of singleton predicate). Let A sing. Then:

1 $\Delta; \Psi \vdash \star^A \Leftarrow A$.

2 If $\Delta; \Psi \vdash M, N \Leftarrow A$ then $\Delta; \Psi \vdash M =_\eta N : A$.

Since we can solve or prune constraints $\Psi \vdash u[\rho] = M : A$ when ρ is a variable substitution, we need to be able to decide whether an arbitrary substitution σ that appears in a constraint $\Psi \vdash u[\sigma] = M : A$ is equivalent to a variable substitution. In essence, we need to decide whether a β-normal form N which is part of σ is equal to a variable x modulo η-equality. In the following, we adapt η-contraction to fulfill this task. Note that the following additional η-like reductions have to be considered.

Law where B sing	condition	justification
$\Psi \vdash \lambda x.\, M\, N \;=_\eta M : \Pi x{:}B.\, C$	$x \notin \mathsf{FV}(M)$	$\Psi, x{:}B \vdash N =_\eta x : B$
$\Psi \vdash (N, \mathsf{snd}\, M) =_\eta M : \Sigma x{:}B.\, C$		$\Psi \vdash N =_\eta \mathsf{fst}\, M \;: B$
$\Psi \vdash (\mathsf{fst}\, M, N) \;=_\eta M : \Sigma x{:}A.\, B$		$\Psi \vdash N =_\eta \mathsf{snd}\, M : [\mathsf{fst}\, M/x]B$

Note that these laws arise at function and pair types that contain a singleton type but are not singleton types.

As a prerequisite to solving constraint $\Psi \vdash u[\sigma] = N : C$, we need to turn σ into a variable substitution ρ. Given $u : B[\Phi]$, we will write this step as $\boxed{\Psi \vdash \sigma \rangle\!\rangle\, \rho \Leftarrow \Phi}$. To this end, we need to check whether a term M of σ is η-equal to a variable, i. e., whether $\Psi \vdash M =_\eta x : A$ for A the type of M. In case A sing, this is trivially true, but also in the other case it is clearly decidable: we could just try the judgement for all $(x{:}A) \in \Psi$. However, this is inefficient; we should be able to compute the variable x that M contracts to in case it contracts to a variable at all. In the following, we develop a judgement such that its application $\Psi \vdash M \rangle\!\rangle\, x \Leftarrow A$ does the job.

Before we dive into the definition, let us consider the example

$$x : \ldots \vdash \lambda y.\, \lambda z.\, x\, (\mathsf{fst}\, y,\ z)\, z =_\eta x : \Pi y{:}(\Sigma \_{:}A.\, \mathbf{1}).\, \Pi z{:}\mathbf{1}.\, B.$$

Our algorithm will first try to establish

$$x : \ldots,\ y : \Sigma \_{:}A.\, \mathbf{1} \vdash \lambda z.\, x\, (\mathsf{fst}\, y,\ z)\, z =_\eta x\, y : \Pi z{:}\mathbf{1}.\, B.$$

However, we cannot η-contract directly here since z occurs in $x\, (\mathsf{fst}\, y,\ z)$. Yet we can get rid of z since it is of singleton type, it is sufficient to derive

$$x : \ldots,\ y : \Sigma \_{:}A.\, \mathbf{1} \vdash x\, (\mathsf{fst}\, y,\ \star)\, \star =_\eta x\, y\, \star : B.$$

We get there by applying both sides to $\star$. The derivation can be closed by observing that

$$x : \ldots,\ y : \Sigma \_{:}A.\, \mathbf{1} \vdash (\mathsf{fst}\, y,\ \star) =_\eta y : \Sigma \_{:}A.\, \mathbf{1}.$$

These thoughts lead to the design of judgement $\boxed{\Psi \vdash M \rangle\!\rangle\, E[x] : A}$ which is directed on non-singleton type A (see Figure 6). It η-contracts term M of type A to a neutral term $E[x]$ with a variable, x, in the head. Ultimately, we are interested in a plain variable $E[x] = x$ as result, but the definition requires a generalization to neutrals.

If the input term is a lambda abstraction $\lambda y.\, M$, necessarily at function type $\Pi y{:}A.\, B$, we distinguish two cases. If A sing then we try to contract the body $[\star^A/y]M$ where we have eliminated the singleton variable y. If the result is an application $E[x]\, N$ with $E[x]$ of the same function type, we know that N is of singleton type, hence, $\lambda y.\, E[x]\, N$ is η-equal to $E[x]$ which we return. Otherwise, if not A sing, then we first try to η-contract the unmodified body M. If the result is an application $E[x]\, N$, and $E[x]$ is of the original

$\boxed{\Psi \vdash M \rangle\!\rangle E[x] \Leftarrow A}$

Input: Ψ, M, A with not A sing. Output: $E[x]$ such that $\Psi \vdash M =_\eta E[x] : A$.

$$\frac{\Psi \vdash E[x] \Leftarrow A}{\Psi \vdash E[x] \rangle\!\rangle E[x] \Leftarrow A}$$

$$\frac{A\ \mathsf{sing} \qquad \Psi \vdash [\star^A/y]M \rangle\!\rangle E[x]\, N \Leftarrow [\star^A/y]B \qquad \Psi \vdash E[x] \Rightarrow \Pi y{:}A.\, B}{\Psi \vdash \lambda y.\, M \rangle\!\rangle E[x] \Leftarrow \Pi y{:}A.\, B}$$

$$\frac{\text{not } A\ \mathsf{sing} \qquad \Psi, y{:}A \vdash M \rangle\!\rangle E[x]\, N \Leftarrow B \qquad \Psi \vdash E[x] \Rightarrow \Pi y{:}A.\, B \qquad \Psi \vdash N =_\eta y : A}{\Psi \vdash \lambda y.\, M \rangle\!\rangle E[x] \Leftarrow \Pi y{:}A.\, B}$$

$$\frac{\Psi \vdash M_1 \rangle\!\rangle \mathsf{fst}\, E[x] \Leftarrow A \text{ unless } A\ \mathsf{sing} \qquad \Psi \vdash M_2 \rangle\!\rangle \mathsf{snd}\, E[x] \Leftarrow [M_1/y]B \text{ unless } B\ \mathsf{sing}}{\Psi \vdash (M_1, M_2) \rangle\!\rangle E[x] \Leftarrow \Sigma y{:}A.\, B}$$

Fig. 6. Type-directed η-contraction.

function type and does not depend on y, we can η-contract the abstraction $\lambda y.\, M$ to $E[x]$ provided argument N is η-equal to variable y.

In case the input term M is a pair (M_1, M_2), it must have pair type $\Sigma y{:}A.\, B$ and at least one of A or B is not a singleton type. We try to contract M_1 to $\mathsf{fst}\, E[x]$ (unless A sing) and M_2 to $\mathsf{snd}\, E[x]$ (unless B sing). If we succeed, we can contract the pair (M_1, M_2) to $E[x]$.

The above example can be written as the following derivation tree, where $(*) = x : \ldots, y : \Sigma\_{:}A.\, \mathbf{1} \vdash (\mathsf{fst}\, y,\, \star) =_\eta y : \Sigma\_{:}A.\, \mathbf{1}$:

$$\cfrac{\cfrac{\cfrac{}{x : \ldots, y : \Sigma\_{:}A.\, \mathbf{1} \vdash x\, (\mathsf{fst}\, y,\, \star)\, \star \rangle\!\rangle x\, (\mathsf{fst}\, y,\, \star)\, \star \Leftarrow [\star/z]B}}{x : \ldots, y : \Sigma\_{:}A.\, \mathbf{1} \vdash \lambda z.\, x\, (\mathsf{fst}\, y,\, z)\, z \rangle\!\rangle x\, (\mathsf{fst}\, y,\, \star) \Leftarrow \Pi z{:}\mathbf{1}.\, B} \qquad (*)}{x : \ldots \vdash \lambda y.\, \lambda z.\, x\, (\mathsf{fst}\, y,\, z)\, z \rangle\!\rangle x \Leftarrow \Pi y{:}(\Sigma\_{:}A.\, \mathbf{1}).\, \Pi z{:}\mathbf{1}.\, B}$$

Lemma 5.2 (η-Contraction to variable). Let A such that not A sing.

1 Soundness: If $\Psi \vdash M \rangle\!\rangle E[x] \Leftarrow A$ then $\Psi \vdash M =_\eta E[x] : A$.
2 Completeness: If $\Psi \vdash M =_\eta E[x] : A$ then $\Psi \vdash M \rangle\!\rangle E'[x] \Leftarrow A$ for some E' with $\Psi \vdash E[x] =_\eta E'[x] : A$. If even $\Psi \vdash M =_\eta x : A$ then $\Psi \vdash M \rangle\!\rangle x \Leftarrow A$.
3 Termination: Given $\Delta; \Psi \vdash M \Leftarrow A$, the query $\Psi \vdash M \rangle\!\rangle ? \Leftarrow A$ terminates.
4 Decidability: Let $\Delta; \Psi \vdash M \Leftarrow A$. Then $\exists x.\, (\Psi \vdash M =_\eta x : A)$ is decided by running $\Psi \vdash M \rangle\!\rangle ? \Leftarrow A$.

Proof. Soundness is an easy induction on the derivation, termination is by induction on the type, decidability is a consequence of soundness, completeness, and termination. Completeness follows from the following inversion properties of η-equality. □

Lemma 5.3 (Inversion of η-equality to neutral term). Let not C sing.

1 If $\Psi \vdash x =_\eta R_0 : C$ then $R_0 = x$ and $\Psi \vdash C =_\eta \Psi(x)$.
2 If $\Psi \vdash R\, M =_\eta R_0 : C$ then $R_0 = R'\, M'$ for some R', M' with $\Psi \vdash R =_\eta R' : \Pi y{:}A.\, B$ for some y, A, B and $\Psi \vdash M =_\eta M' : A$ and $\Psi \vdash C =_\eta [M/y]B$.

3 If $\Psi \vdash \mathsf{fst}\, R =_\eta R_0 : C$ then $R_0 = \mathsf{fst}\, R'$ with $\Psi \vdash R =_\eta R' : \Pi y{:}C.\, D$ for some y, D.
4 If $\Psi \vdash \mathsf{snd}\, R =_\eta R_0 : C$ then $R_0 = \mathsf{snd}\, R'$ and $\Psi \vdash R =_\eta R' : \Pi y{:}A.\, B$ for some y, A, B with $\Psi \vdash C =_\eta [\mathsf{fst}\, R/y]B$.
5 If $\Psi \vdash \lambda y.\, M =_\eta R : C$ then $C = \Pi y{:}A.\, B$ and $\Psi, y{:}A \vdash M =_\eta R\, y : B$.
6 If $\Psi \vdash \lambda y.\, M =_\eta R : \Pi y{:}A.\, B$ with A sing then $\Psi \vdash [\star^A/y]M =_\eta R\, \star^A : [\star^A/y]B$.
7 If $\Psi \vdash (M_1, M_2) =_\eta R : C$ then $C = \Sigma y{:}A.\, B$ and $\Psi \vdash M_1 =_\eta \mathsf{fst}\, R : A$ and $\Psi \vdash M_2 =_\eta \mathsf{snd}\, R : [\mathsf{fst}\, M_1/y]B$.

Proof. Statements 1–4 hold since η-equality does not modify neutrals of non-singleton type, only λs or pairs. Statements 5–7 are easy applications of congruence rules and β-normalization. For instance, statement 5 arises from weakening and application to variable y (with subsequent renormalization). □

Based on judgement $\Psi \vdash M \rangle\!\rangle\, x \Leftarrow A$ we define η-contraction of a substitution σ to a variable substitution ρ as judgement $\boxed{\Psi \vdash \sigma \rangle\!\rangle\, \rho \Leftarrow \Phi}$ in the obvious way.

In the following, we amend the unification algorithm to handle unit types. *Local simplification* (see Fig. 3) is extended by a new transition that deletes boring constraints, i. e., those at singleton types:

"Decomposition" of singletons (1)
$\Phi \vdash M = N : A \;\mapsto_{\mathsf{d}}\; \top\;$ if A sing

Constraints can be freed of variables in subterms of singleton type. Those variables could otherwise trip up the occurs check, pruning, or inverse substitution. Let M $\mathsf{sing\_free}$ express that all subterms of M of type A sing are $\star^A$.

Eliminating singleton subterms (2)
$\Phi \vdash u[\sigma] = M : C \quad \mapsto_{\mathsf{e}} \quad \Phi \vdash u[\sigma] = M' : C$
if $\Phi \vdash M =_\eta M' : C$ and M' $\mathsf{sing\_free}$ but not M $\mathsf{sing\_free}$

The occurs check needs to be restricted to rhss that have no variables in subterms of singleton types, as those are eliminateable by η.

Failing occurs check only when M $\mathsf{sing\_free}$ (3)
$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = M : C \mapsto \bot$ if $\mathsf{FV}^{\mathsf{rig}}(M) \not\subseteq \rho$
$\Delta \Vdash \mathcal{K} \wedge \Psi \vdash u[\rho] = M : C \mapsto \bot$ if $M = M'\{u[\xi]\}^{\mathsf{srig}} \neq u[\xi]$

In an implementation, the elimination of singleton subterms could be incorporated, together with the occurs check, into pruning, to avoid multiple traversals of the rhs.

For the following transitions, assume $(u : A[\Phi]) \in \Delta$ active. The η-contraction transitions $\mapsto_{\mathsf{e}}$ are replaced by a transition that turns a substitution into a variable substitution.

η-Contraction (4)
$\Psi \vdash u[\sigma] = N : A \;\mapsto_{\mathsf{e}}\; \Psi \vdash u[\rho] = N : A\;$ if $\;\Psi \vdash \sigma \rangle\!\rangle\, \rho \Leftarrow \Phi$.

Meta variables of singleton types can be solved on the spot:

Solving singleton metas (5)
$\Delta \Vdash \mathcal{K} \quad \mapsto \quad \Delta \Vdash \mathcal{K} + \Phi \vdash u \leftarrow \star^A : A \quad$ if $\;A$ sing.

This completes the extension to singleton types. We have taken care of singleton types in all relevant positions:

1 In the type of a constraint: (1) and (3).

2 In the terms of a constraint: (2) and (4).
3 In the type of a meta variable (5).

We could also eliminate singleton variables from contexts, via the two following transitions:

Eliminating singleton variables (6)
$\Phi_1, x{:}A, \Phi_2 \vdash M = N : C \;\mapsto_{\mathsf{p}}\; \Phi_1, [\tau]\Phi_2 \vdash [\tau]M = [\tau]N : [\tau]C$ if A sing
where $\tau = [\star^A/x]$.

Pruning singleton variables (7)
$\Delta \Vdash \mathcal{K} \;\mapsto\; \big(\Delta, v : [\Phi_1, [\star^B/x]\Phi_2]([\star^B/x]A) \Vdash \mathcal{K}\big) \,+\, (\Phi \vdash u \leftarrow v[\mathsf{wk}_{\Phi_1,\Phi_2}] : A)$
if $\Phi = \Phi_1, x{:}B, \Phi_2$ and B sing.

However, singleton variables in contexts are harmless as long as we apply (2) to eliminate their occurrences in constraints.

6. Related work

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 extends prior work by Reed [2009b] to handle Σ-types. Previously, Elliot [1990] 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 [Schack-Nielsen and Schürmann, 2010, Norell, 2007], 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 [1996] 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 [1998] 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.

7. 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 [Dowek et al., 1996], but is also less ambitious than the algorithm proposed by Reed [2009b]. We have extended higher-order pattern unification to handle Σ-types; this has been an open problem so far, yet it is of practical relevance:

1 In LF-based systems such as Beluga, Twelf or Delphin, a limited form of Σ-types arises due to context blocks: Σ-types are used to introduce several assumptions simultaneously. For Beluga, the second author has implemented the flattening of context blocks and it works well in type reconstruction [Pientka, 2013].

2 In dependently typed languages such as Agda, Σ-types, or, more generally, record types, are commonly used. However, implementations of unification in these systems have traditionally not supported records. McBride [2010, p. 6] gives a practical example where the unification problem $T\,(\mathsf{fst}\,\gamma)\,(\mathsf{snd}\,\gamma) = T'\,\gamma$ appears. Using the techniques presented in this paper, the first author has extended Agda's unification algorithm to solve those kinds of problems.

Correctness of our unification constraint solver is proven using *typing modulo* Reed [2009b]. This is possible since we have no constraints on the type level and we are dealing with terms whose normalization via hereditary substitutions can be defined by recursion on their type. Even in the presence of unsolvable constraints, which lead to ill-typed terms, normalization is terminating. This does not scale to Agda which has large eliminations and unification on the type level; there, ill-typed terms may lead to divergence of type reconstruction. A solution has been described by Norell [2007]: unsolved constraints block normalization, thus guaranteeing termination of the unification algorithm. The idea has been implemented in Agda 2 and been extended to Σ-types and the unification rules described in this article.

Acknowledgments. We thank Jason Reed for his insightful work and his explanations given via email. Thanks to Conor McBride for in-depth comments on this text and his suggestions for improvement. We also acknowledge the anonymous referees who have given constructive critique on a previous version of this article.

References

A. Abel and B. Pientka. Higher-order dynamic pattern unification for dependent types and records. In C.-H. L. Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011, Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 10–26. Springer, 2011. ISBN 978-3-642-21690-9. .

AgdaTeam. The Agda Wiki, 2018.

I. Cervesato and F. Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.

G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In *Joint International Conference Logic Programming*, pages 259–273. MIT Press, 1996.

D. Duggan. Unification with extended patterns. *Theoretical Computer Science*, 206(1-2): 1–50, 1998.

C. Elliott. *Extensions and Applications of Higher-Order Unification.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1990.

R. Fettig and B. 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)*, LNCS 1103, pages 347–361. Springer, 1996. ISBN 3-540-61464-8.

A. Gacek. The Abella interactive theorem prover (system description). In *4th International Joint Conference on Automated Reasoning*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 154–161. Springer, 2008.

H. Goguen. Justifying algorithms for $\beta\eta$ conversion. In *8th Int. Conf. on Foundations of Software Science and Computational Structures (FoSSaCS'05)*, LNCS 3441, pages 410–424. Springer, 2005. ISBN 3-540-25388-2.

W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

R. Harper and D. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 17(4–5):613–673, July 2007.

C. McBride. Outrageous but meaningful coincidences: Dependent type-safe syntax and evaluation. In *ACM SIGPLAN Workshop on Genetic Programming (WGP'10)*, pages 1–12. ACM, 2010.

S. Michaylov and F. Pfenning. An empirical study of the runtime behavior of higher-order logic programs. In D. Miller, editor, *Proceedings of the Workshop on the lambda Prolog Programming Language*, pages 257–271. University of Pennsylvania, 1992. Available as Technical Report MS-CIS-92-86.

D. Miller. Unification of simply typed lambda-terms as logic programming. In *Eighth International Logic Programming Conference*, pages 255–269. MIT Press, 1991.

G. Nadathur and N. Linnell. Practical higher-order pattern unification with on-the-fly raising. In *21st International Conference on Logic Programming (ICLP'05)*, Lecture Notes in Computer Science (LNCS 3668), pages 371–386. Springer-Verlag, 2005. ISBN 3-540-29208-X, 978-3-540-29208-1. .

G. Nadathur and D. J. Mitchell. System description: Teyjus – a compiler and abstract machine based implementation of Lambda Prolog. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 287–291, Trento, Italy, July 1999. Springer-Verlag LNCS.

A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

U. Norell. *Towards a Practical Programming Language Based on Dependent Type Theory.* PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, Sept. 2007.

F. Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, 5-8 June, 1989*, pages 313–322. IEEE Computer Society Press, 1989. ISBN 0-8186-1954-6. .

F. Pfenning and C. Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In *16th International Conference on Automated Deduction (CADE-16)*, LNAI 1632, pages 202–206. Springer, 1999.

B. Pientka. *Tabled higher-order logic programming.* PhD thesis, Department of Computer Science, Carnegie Mellon University, 2003. CMU-CS-03-185.

B. Pientka. An insider's look at LF type reconstruction: Everything you (n)ever wanted to know. *Journal of Functional Programming*, 1(1–37), 2013.

B. Pientka and A. Cave. Inductive Beluga:Programming Proofs (System Description). In A. P. Felty and A. Middeldorp, editors, *25th International Conference on Automated Deduction (CADE-25)*, Lecture Notes in Computer Science (LNCS 9195), pages 272–281. Springer, 2015.

B. Pientka and J. Dunfield. Beluga: a framework for programming and reasoning with deductive systems (System Description). In *5th International Joint Conference on Automated Reasoning (IJCAR'10)*, LNAI 6173, pages 15–21, 2010.

J. Reed. *A Hybrid Logical Framework*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2009a.

J. Reed. Higher-order constraint simplification in dependent type theory. In *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP'09)*, 2009b.

A. Schack-Nielsen and C. Schürmann. Curry-style explicit substitutions for the linear and affine lambda calculus. In J. Giesl and R. Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2010. ISBN 978-3-642-14202-4. .

N. Shankar. Inference systems for logical algorithms. volume 3821 of *Lecture Notes in Computer Science*, pages 60–78. Springer, 2005. ISBN 3-540-30495-9. .

K. Watkins, I. Cervesato, F. Pfenning, and D. Walker. A concurrent logical framework I: Judgements and properties. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2003.