Extending Higher-Order Unification to support Proof Irrelevance*

Jason Reed

Carnegie Mellon University, Pittsburgh, PA 15213, USA
jcreed@cs.cmu.edu

Abstract. As theories and proofs in logical frameworks become larger, careful control over what information within them can safely be omitted or erased becomes useful for efficient implementation. The notion of proof irrelevance provides exactly this control, but requires existing algorithms used in logical frameworks, in particular higher-order pattern unification, to be extended to accommodate the richer type theory. We describe this extended algorithm, whose presentation is simplified by making use of recent developments in explaining unification metavariables as modal variables, which obviates the need for full explicit substitutions.

1 Introduction

1.1 Proof Irrelevance

Proof irrelevance is the treatment of all proofs of the same proposition as equal. The type system introduced by Pfenning [7] defines (in addition to an intentional equality, with which we are not concerned here) *irrelevant* versions of the ordinary typing and equality judgments. A characteristic feature of both these judgments are that they are allowed to make use of *irrelevant hypotheses* in the context, which are intuitively assumptions that some proof of a proposition (i.e. term at a type) exists that are provided only under a promise that the consumer of the assumption does not care *which* proof it is. The irrelevant version of the equality judgment is distinguished by being trivial, holding if merely the terms being compared are both well-typed. Since there is a new kind of hypothesis in the definition of contexts, there is a new function type as well, one with an irrelevant domain.

As an example of the expressivity of these functions, suppose we want to represent a lambda calculus which has a binder $\lambda^{\geq 1}$ which requires the variable it binds to occur at least once. We could try doing this by beginning with the usual higher-order abstract syntax [8] encoding of the untyped lambda calculus

$$\begin{aligned}&\mathsf{tm} : \mathsf{type}\\&\mathsf{app} : \mathsf{tm} \to (\mathsf{tm} \to \mathsf{tm})\\&\mathsf{lam} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm}\end{aligned}$$

* This work has been partially supported by NSF Grant CCR-9988281 *"Logical and Meta-Logical Frameworks"*. Thanks are owed to Frank Pfenning and Kaustuv Chaudhuri for many useful comments and corrections.

defining a predicate $\mathsf{occurs} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{type}$ which is inhabited by proofs that a term closed except for one free variable actually uses it, and declaring a constant

$$\mathsf{olam} : \Pi t{:}(\mathsf{tm} \to \mathsf{tm}).\, \mathsf{occurs}\, t \to \mathsf{tm}$$

to represent the binder $\lambda^{\geq 1}$. This encoding, however, is not adequate, in the sense that there are generally multiple representation-language terms which represent an object-language term. This is because, in any natural logic program implementing the predicate occurs, each particular occurrence of the variable acts as a different witness of the fact that it occurs at least once, and so there are potentially many proofs of $\mathsf{occurs}\, t$.

We want, however, to equate all terms constructed using olam that differ only in which proof they use, for the proof ought to have no effect on the equational theory of terms — it only matters *that* the proofs exist. That is, we want an irrelevant function arrow $\to^{\mathsf{i}}$, and we should declare[1] instead

$$\mathsf{olam} : \Pi t{:}(\mathsf{tm} \to \mathsf{tm}).\, \mathsf{occurs}\, t \to^{\mathsf{i}} \mathsf{tm}$$

and the encoding would then be adequate.

Moreover, recent work with large safety proofs in a metalogical style arising from grid computing applications [2] suggests that significant space and time savings (as alluded to in [7]) stand to be made in them without sacrificing soundness if only this sort of function type were available in the implementation of the logical framework used. These would allow many intermediate proof objects to be 'forgotten' during checking, because the type system guarantees that their exact identity is not necessary after their existence has been verified. This metalogical approach relies heavily, however, on the logic programming interpretation of type family declarations, which in turn relies on algorithms for unification, as well as mode checking and totality checking [12]. Revised algorithms are therefore necessary for the revised underlying type theory. We focus on the first of these algorithms as an initial step along this path, the revision of higher-order pattern unification in a dependent type theory similar to LF [5].

1.2 Higher-Order Pattern Unification

Using explicit substitutions [1], higher-order unification can be reduced to a first-order equational unification problem [3]. Though higher-order unification is in fact undecidable in general, it is decidable for the pattern subset [4]. This decidable subset can be also solved dynamically, putting off non-pattern equations in the hope that later instantiations will make them pattern. This constraint simplification algorithm turns to be quite useful in practice, solving many systems of equations which do not initially consist of only patterns.

We had previously tried [10] to start with the algorithm as presented by Dowek et al. but it is not clear how to achieve right marriage of proof irrelevance and explicit substitutions. Either additional equations or additional rewrite rules

[1] This notation is merely suggestive. The syntax is made precise in the next section.

are necessary to establish the triviality of equality 'at irrelevant types,' but achieving and working out the desired confluence and termination properties seems less than trivial.

Fortunately, these difficulties can be sidestepped entirely. The use of substitutions as full first-class citizens in the language can be eliminated in favor of limiting their role to only occurring applied to metavariables open to instantiation, and conversely requiring metavariables to only occur immediately under a substitution. This approach [9] involves a typing discipline derived from a modal logic of (relative) validity, and reflects faithfully aspects of the actual implementation of unification variables.

2 Language

2.1 Syntax

We work throughout with a deBruijn-index style version of LF, augmented with proof irrelevance [7], and relative validity variables [9] as described above. While the use of deBruijn indices is at a lower level of description than is strictly necessary, the precision which they provide is useful for concisely defining and reasoning about the auxiliary notions necessary for the algorithm presented below. It also has the useful side effect of highlighting the distinction between ordinary variables (represented as indices) and unification variables (described in terms of modal variables).

The syntax is given by

$$\begin{array}{rl}
\text{Status } \mu ::= & \mathsf{r} \mid \mathsf{i} \\
\text{Kinds } K ::= & \mathsf{type} \mid \Pi^{\mu}_{A} K \\
\text{Families } A, B ::= & a \mid A \diamond^{\mu} M \mid \Pi^{\mu}_{A} B \\
\text{Objects } M, N ::= & c \mid M \diamond^{\mu} N \mid \lambda^{\mu}_{A} M \mid \mathsf{n} \mid u[\sigma] \\
\text{Substitutions } \sigma ::= & \cdot \mid M \cdot^{\mu} \sigma \\
\text{Contexts } \Gamma, \Psi ::= & \cdot \mid \Gamma \cdot^{\mu} A \\
\text{Modal Contexts } \Delta ::= & \cdot \mid \Delta, u :: (\Psi \vdash A)
\end{array}$$

Constant families are typically written a, b, constant objects c, modal variables u, v, and deBruijn indices n.

For uniformity of notation, we use superscript annotations (generally μ) on various pieces of syntax indicating whether they are relevant (r) or irrelevant (i) versions. On binders (λ^{μ}_{A}, Π^{μ}_{A}) the μ indicates whether the bound variable is 'relevantly' or 'irrelevantly' at the appropriate type, that is, whether the function or function type respectively can or cannot distinguish between different arguments passed to it. Function application ($M \diamond^{\mu} N$, $A \diamond^{\mu} M$) is written with the non-blank symbol $\diamond$ so that it can carry an annotation μ, to bring the elimination forms for function types in correspondence with the introduction forms. That is, for $M \diamond^{\mathsf{r}} N$ to be well-typed, M should have type $\Pi^{\mathsf{r}}_{A}.B$ for some A, B, and for $M \diamond^{\mathsf{i}} N$ to be well-typed, M should have a type of the shape $\Pi^{\mathsf{i}}_{A}.B$. Conses $\Gamma \cdot^{\mu} A$ of a variable at type A onto a context Γ have a μ to indicate whether the variable is a relevant or irrelevant hypothesis. Contexts are written left-to-right (that is, the deBruijn index $\mathsf{1}$ refers to a variable at the rightmost type in the context) so that, as customary, types only depend on other types on their left. Conses $M \cdot^{\mu} \sigma$ of terms M onto substitutions σ carry a μ to indicate whether the M should be typed (and dealt with respect to equality) relevantly or irrelevantly.

We abbreviate $:^{\mathsf{r}}$ by $:$ and $:^{\mathsf{i}}$ by $\div$ to match the notation of [7].

The 'irrelevant arrow' used in the example in the introduction becomes a Π by defining the syntactic sugar $A \to^{\mu} B = \Pi^{\mu}_{A}.B$ where B does not actually depend on A, a simple extension of that typically made for $\to$ in dependent type theories. The full typing of olam from the example in the introduction, in the syntax developed thus far, is

$$\mathsf{olam} : \Pi^{\mathsf{r}}_{\Pi^{\mathsf{r}}_{\mathsf{tm}}\,\mathsf{tm}} \Pi^{\mathsf{i}}_{\mathsf{occurs} \diamond^{\mathsf{r}} \mathsf{1}}\,\mathsf{tm}$$

The syntax $u :: (\Psi \vdash A)$ [9] indicates a *modal variable* u which stands in place of a term which has type A in the context Ψ. The variable u is 'relatively closed' (logically, represents a hypothesis of the 'relative validity' of A relative to Ψ) in that it cannot refer to any variables in any context but Ψ.

2.2 Definitions

Since the presentation is nameless, we sketch the definition of shifting and substitution, only for the object level. The definitions given should not be surprising, as they are essentially the same as they would be if irrelevance were absent, except that all annotations μ are simply preserved. The definitions at the family and kind levels are straightforward.

The operation $\uparrow_m$ increments all indices $\geq m$. Note that this is merely an operation on terms, not a full-fledged substitution. It is defined by

$$\uparrow_m c = c \qquad \uparrow_m (M \diamond^{\mu} N) = \uparrow_m M \diamond^{\mu} \uparrow_m N \qquad \uparrow_m (\lambda^{\mu}_{A} M) = \lambda^{\mu}_{\uparrow_m A}(\uparrow_{m+1} M)$$

$$\uparrow_m \mathsf{n} = \begin{cases} \mathsf{n+1} & \text{if } n \geq m; \\ \mathsf{n} & \text{otherwise.} \end{cases} \qquad \uparrow_m (u[\sigma]) = u[\uparrow_m \sigma]$$

$$\uparrow_m (\cdot) = \cdot \qquad \uparrow_m (M \cdot^{\mu} \sigma) = \uparrow_m M \cdot^{\mu} \uparrow_m \sigma$$

We abbreviate $\uparrow_1$ by just $\uparrow$. Ordinary substitution $[\sigma]M$ (again merely an operation on terms) is defined by

$$[\sigma]c = c \qquad [\sigma](M \diamond^{\mu} N) = [\sigma]M \diamond^{\mu} [\sigma]N \qquad [\sigma]\lambda^{\mu}_{A} M = \lambda^{\mu}_{[\sigma]A}[\mathsf{1} \cdot^{\mu} \uparrow\sigma]M$$

$$[M \cdot^{\mu} \sigma]\mathsf{1} = M \qquad [M \cdot^{\mu} \sigma]\mathsf{n+1} = [\sigma]\mathsf{n} \qquad [\sigma](u[\tau]) = u[[\sigma]\tau]$$

$$[\sigma](\cdot) = \cdot \qquad [\sigma](M \cdot^{\mu} \tau) = [\sigma]M \cdot^{\mu} [\sigma]\tau$$

Following [9], the *operation* of substitution is written as a prefix operator to distinguish the substitution attached to occurrences of modal variables $u[\sigma]$, which is written postfix.

The introduction of modal variables carries with it a novel substitution principle. The substitution operation $[\![P/u]\!]$ of a (closed, relative to the context Ψ) term P for the modal variable $u :: (\Psi \vdash A)$ for some context Ψ and type A, is defined by

$$[\![P/u]\!]c = c \qquad [\![P/u]\!](M \diamond^{\mu} N) = [\![P/u]\!]M \diamond^{\mu} [\![P/u]\!]N$$

$$[\![P/u]\!]\lambda^{\mu}_{A} M = \lambda^{\mu}_{[\![P/u]\!]A}[\mathsf{1} \cdot^{\mu} \uparrow\sigma]M \qquad [\![P/u]\!]\mathsf{n} = \mathsf{n}$$

$$[\![P/u]\!](u[\tau]) = [[\![P/u]\!]\tau]P$$

$$[\![P/u]\!](v[\tau]) = v[[\![P/u]\!]\tau] \qquad (\text{for } u \neq v)$$

$$[\![P/u]\!](\cdot) = \cdot \qquad [\![P/u]\!](M \cdot^{\mu} \tau) = [\![P/u]\!]M \cdot^{\mu} [\![P/u]\!]\tau$$

Since P can only refer to variables in the context Ψ, variable capture is impossible, and this substitution can be (and in practice is) carried out in the implementation by simple in-place update.

We use θ for simultaneous modal substitutions $P_1/u_1, P_2/u_2, \ldots, P_n/u_n$. We say $\Delta' \vdash \theta : \Delta$ if θ is a substitution for the variables $u_i :: (\Psi_i \vdash A_i)$ in Δ and $\Delta'; \Psi_i \vdash P_i : A_i$ for all i. The relevant substitution property is

Lemma 1. *If $\Delta; \Gamma \vdash M : A$ and $\Delta' \vdash \theta : \Delta$, then $\Delta'; [\![\theta]\!]\Gamma \vdash [\![\theta]\!]M : [\![\theta]\!]A$.*

which is easily established by induction. The identity substitution is defined by

$$\mathsf{id}_{\cdot} = \cdot \qquad \mathsf{id}_{\Gamma \cdot^{\mu} A} = \mathsf{1} \cdot^{\mu} \uparrow(\mathsf{id}_{\Gamma})$$

We often omit the subscript when it is clear from context.

The typing rules require a context promotion operation $—^{\oplus}$, defined by

$$\cdot^{\oplus} = \cdot \qquad (\Gamma \cdot^{\mu} A)^{\oplus} = (\Gamma^{\oplus} \cdot^{\mathsf{r}} A)$$

which promotes all irrelevant assumptions to relevant.

2.3 Judgments

The central judgments are

$$\begin{array}{rl}
\vdash \Delta\ \mathsf{mctx} & \Delta \text{ is a valid modal context} \\
\Delta \vdash \Gamma\ \mathsf{ctx} & \Gamma \text{ is a valid context} \\
\Delta; \Gamma \vdash M = N :^{\mu} A & M \text{ and } N \text{ are equal at type } A \\
\Delta; \Gamma \vdash M :^{\mu} A & M \text{ has type } A \\
\Delta; \Gamma \vdash \sigma : \Psi & \sigma \text{ has type } \Psi
\end{array}$$

There are a handful of other, analogous judgments at the family and kind level, which we again omit for brevity.

The judgments $\Delta\ \mathsf{mctx}$ and $\vdash \Gamma\ \mathsf{ctx}$, as common for dependent type theories, insure that every type in the context Γ (resp. modal context Δ) is actually constructed correctly from previous context variables. We omit the exact rules defining them.

The equality and typing judgments each have a μ annotation to indicate whether the equality or typing, respectively, takes place in a relevant or irrelevant way. The following sections detail their effect.

2.4 Object Typing

The critical rule for proof irrelevance is the sole rule which has an irrelevant typing judgment as its conclusion:

$$\frac{\Delta; \Gamma^{\oplus} \vdash M : A \qquad \Delta; \Gamma \vdash A : \mathsf{type}}{\Delta; \Gamma \vdash M \div A}$$

which allows (via the promotion operation defined above) irrelevant assumptions in Γ to be used to show that $M \div A$, as long as A is a valid type in Γ unpromoted. ([11] treats why this second premise is necessary)

The other typing rules are mostly straightforward:

$$\frac{c : A \in \Sigma}{\Delta; \Gamma \vdash c : A}$$

$$\frac{\Delta; \Gamma \vdash M : \Pi^{\mu}_{A} B \qquad \Delta; \Gamma \vdash N :^{\mu} A}{\Delta; \Gamma \vdash M \diamond^{\mu} N : [N \cdot^{\mu} \mathsf{id}_{\Gamma}]B} \qquad \frac{\Delta; \Gamma \cdot^{\mu} A \vdash M : B}{\Delta; \Gamma \vdash \lambda^{\mu}_{A} M : \Pi^{\mu}_{A} B}$$

$$\frac{}{\Delta; \Gamma \cdot^{\mathsf{r}} A \vdash \mathsf{1} : A} \qquad \frac{\Delta; \Gamma \vdash \mathsf{n} : B}{\Delta; \Gamma \cdot^{\mu} A \vdash \mathsf{n+1} : \uparrow B}$$

$$\frac{\Delta; \Gamma \vdash M : A \qquad \Delta; \Gamma \vdash A = B : \mathsf{type}}{\Delta; \Gamma \vdash M : B}$$

The rule for typing uses of modal variables $u :: (\Psi \vdash A)$ is

$$\frac{\Delta, u :: (\Psi \vdash A), \Delta'; \Gamma \vdash \sigma : \Psi}{\Delta, u :: (\Psi \vdash A), \Delta'; \Gamma \vdash u[\sigma] : [\sigma]A}$$

Intuitively, it expresses the fact that u requires (and requires no more than) a justification of Ψ to be usable; if a substitution for the variables of Ψ valid in the contexts $\Delta, u :: (\Psi \vdash A), \Delta', \Gamma$ can be found, it can be attached to u to form an object of type $[\sigma]A$.

Substitutions themselves are typed by

$$\frac{}{\Delta; \Gamma \vdash \cdot : \cdot}$$

$$\frac{\Delta; \Gamma \vdash M :^{\mu} [\sigma]A \qquad \Delta; \Gamma \vdash \sigma : \Psi}{\Delta; \Gamma \vdash M \cdot^{\mu} \sigma : \Psi \cdot^{\mu} A}$$

2.5 Object Equality

For brevity we only mention a few equality rules relevant to proof irrelevance. The others are largely similar to those in the original definition of LF.

The first is

$$\frac{\Delta; \Gamma^{\oplus} \vdash M : A \qquad \Delta; \Gamma^{\oplus} \vdash M' : A \qquad \Delta; \Gamma \vdash A : \mathsf{type}}{\Delta; \Gamma \vdash M = M' \div A}$$

which gives the meaning of proof irrelevance: M and M' are equal at $\div A$ if they are both merely well-typed.

This 'irrelevant equality' is made use of in the rule

$$\frac{\Delta; \Gamma \vdash M = M' : \Pi^{\mu}_{A} B \qquad \Delta; \Gamma \vdash N = N' :^{\mu} A}{\Delta; \Gamma \vdash M \diamond^{\mu} N = M' \diamond^{\mu} N' : [N \cdot^{\mu} \mathsf{id}_{\Gamma}]B}$$

which means in particular that $M \diamond^{\mathsf{i}} N = M \diamond^{\mathsf{i}} N'$, even if $N \neq N'$, as long as both N and N' are well-typed.

Continuing with the example from the introduction, suppose we have some object $t : \Pi^{\mathsf{r}}_{\mathsf{tm}}\, \mathsf{tm}$ (representing an object-language term with one free variable) and two different proofs $p_1, p_2 : \mathsf{occurs} \diamond^{\mathsf{r}} t$. Now we should have

$$\Delta; \Gamma \vdash \mathsf{olam} \diamond^{\mathsf{r}} t \diamond^{\mathsf{i}} p_1 = \mathsf{olam} \diamond^{\mathsf{r}} t \diamond^{\mathsf{i}} p_2 : \mathsf{tm}$$

and indeed we do; to derive this judgment we must show

$$\Delta; \Gamma \vdash \mathsf{olam} \diamond^{\mathsf{r}} t = \mathsf{olam} \diamond^{\mathsf{r}} t : \Pi^{\mathsf{r}}_{\mathsf{occurs} \diamond^{\mathsf{r}} t}\, \mathsf{tm}$$

which follows by a reflexivity lemma, and

$$\Delta; \Gamma \vdash p_1 = p_2 \div \mathsf{occurs} \diamond^{\mathsf{r}} t$$

which holds precisely because of the former equality rule above.

Substitution equality (required, incidentally for equality of modal variable uses $u[\sigma]$) is defined by

$$\frac{}{\Delta; \Gamma \vdash \cdot = \cdot : \cdot}$$

$$\frac{\Delta; \Gamma \vdash M = M' :^{\mu} [\sigma]A \qquad \Delta; \Gamma \vdash \sigma = \sigma' : \Psi}{\Delta; \Gamma \vdash M \cdot^{\mu} \sigma = M' \cdot^{\mu} \sigma' : \Psi \cdot^{\mu} A}$$

Notice that in the case of $\mu = \mathsf{i}$, at irrelevant conses $\cdot^{\mathsf{i}}$ the consed terms M, M' are only required to be irrelevantly equal ($\Delta; \Gamma \vdash M = M' \div [\sigma]A$), that is, merely both well-typed.

3 Unification

3.1 Pattern Substitutions

Pattern substitutions [6, 4], ordinarily substitutions consisting of only distinct bound variables, are distinguished by being injective, that is, there is at most one P such that $[\sigma]P = M$ if σ is pattern. This property makes decidable the subset of unification problems where all terms being unified are *higher-order patterns*, that is, terms where all metavariables occur under pattern substitutions.

We define the judgment $I \vdash \sigma\, \mathsf{pat}$ ("σ is a pattern substitution using variables I") for I a list of natural numbers and σ a substitution. This is a simple generalization of the ordinary definition of pattern substitution. The difference is that here we allow arbitrary terms to occur at irrelevant positions.

$$\frac{}{\vdash \cdot\ \mathsf{pat}}$$

$$\frac{I \vdash \sigma\ \mathsf{pat}}{I \vdash M \cdot^{\mathsf{i}} \sigma\ \mathsf{pat}} \qquad \frac{I \vdash \sigma\ \mathsf{pat} \qquad n \notin I}{I, n \vdash \mathsf{n} \cdot^{\mathsf{r}} \sigma\ \mathsf{pat}}$$

A term in context $\Delta; \Gamma \vdash M : A$ is said to be an *atomic pattern* if every subterm of M of the form $u[\sigma]$ is such that $u :: (\Psi \vdash B) \in \Delta$ with B an atomic (that is, not a function) type, and $I \vdash \sigma\ \mathsf{pat}$ for some I. Assuming all modal variables are atomic incurs no loss, however. The process of *lowering* unification variables present in the given problem to atomic type and *raising* them again before presenting the answer to the user is well-known, and in fact is concisely justified by the modal variable framework [9].

3.2 Inversion

The fact that pattern substitutions are injective has a constructive proof, namely an algorithm (see Figure 1) for computing the inverse when it exists. It is mutually recursive with the definition of the restriction substitution $\xi \,|\, \sigma$.

The correctness of inversion amounts to the two lemmas stated below. Since inversion is computed in the process of unification, however, it can generate new equations as necessary. There is, therefore, also a sort of conceptual mutual recursion going on with the definition of the unification algorithm itself. The statement and style of proof of these lemmas could be separated from the language of unification, but not without some complication. We therefore assume familiarity with typical reasoning about equation-rewriting unification algorithms and speak of terms being known or being shown to be equal 'modulo' the other equations in the system or being newly introduced. See the following section for details of what it means for variables to be added to Δ_B.

Lemma 2. *If $[\sigma]^{-1}M$ exists, then $[\sigma][\sigma]^{-1}M = M$, modulo the additional equations generated, if any.*

Proof. By induction on M. We show just a few representative cases.

For **NI**, we must show $\mathsf{n} = [M \cdot^{\mathsf{i}} \sigma](\uparrow([\sigma]^{-1}\mathsf{n}))$. but it can be shown that $[M \cdot^{\mu} \sigma](\uparrow N) = [\sigma]N$ for any M, N, μ, σ, (cf. the $\lambda\sigma$ [1] rewrite rule **ShiftCons**) so we apply the induction hypothesis to $[\sigma][\sigma]^{-1}\mathsf{n}$.

For **A**$-$, use induction hypothesis and the fact that $M \diamond^{\mathsf{i}} N = M \diamond^{\mathsf{i}} v[\sigma]$.

In the case of **V**$-$ we must apply the induction hypothesis to $[\xi]\xi'$, but by inspection of the rules defining the restriction operation, it can be seen that $[\xi](\xi \,|\, \sigma)$ is just a substitution consisting of a subsequence of the terms in ξ, and so is no bigger than ξ. So we have $[\sigma](u'[[\sigma]^{-1}([\xi]\xi']]) = u'[[\sigma][\sigma]^{-1}([\xi]\xi')]$, which by the induction hypothesis is equal to $u'[[\xi]\xi'] = [\xi](u'[\xi']) = [\xi](u[\mathsf{id}]) = u[\xi]$. It is also easy to show that $[\xi](\xi \,|\, \sigma)$ always exists, by the definition of restriction. ∎

C	$[\sigma]^{-1}c = c$		
NI	$[M \cdot^{\mathsf{i}} \sigma]^{-1}\mathsf{n} = \uparrow([\sigma]^{-1}\mathsf{n})$		
N≠	$[m \cdot^{\mathsf{r}} \sigma]^{-1}\mathsf{n} = \uparrow([\sigma]^{-1}\mathsf{n})$	if $m \neq n$	
N=	$[n \cdot^{\mathsf{r}} \sigma]^{-1}\mathsf{n} = 1$		
E	$[\cdot]^{-1}\mathsf{n}$	fails	
L	$[\sigma]^{-1}(\lambda^{\mu}_{A} M) = \lambda^{\mu}_{[\sigma]^{-1}A}[1 \cdot^{\mu} \uparrow\sigma]^{-1}M$		
A	$[\sigma]^{-1}(M \diamond^{\mu} N) = [\sigma]^{-1}M \diamond^{\mu} [\sigma]^{-1}N$	if $[\sigma]^{-1}N$ exists	
A−	$[\sigma]^{-1}(M \diamond^{\mu} N) = [\sigma]^{-1}M \diamond^{\mu} v[\mathsf{id}]$	if not, for v added to Δ_B	
V+	$[\sigma]^{-1}(u[\xi]) = u[[\sigma]^{-1}\xi]$	if $[\sigma]^{-1}\xi$ exists	
V−	$[\sigma]^{-1}(u[\xi]) = u'[[\sigma]^{-1}([\xi]\xi')]$	if not, adding the equation that $u[\mathsf{id}] = u'[\xi']$ where $\xi' = \xi \mid \sigma$, for u' added to Δ_B	
SN	$[\sigma]^{-1}(\cdot) = \cdot$		
SC	$[\sigma]^{-1}(M \cdot^{\mu} \xi) = [\sigma]^{-1}M \cdot^{\mu} [\sigma]^{-1}\xi$		
R+	$(M \cdot^{\mu} \xi) \mid \sigma = 1 \cdot^{\mu} (\uparrow(\xi \mid \sigma))$	if $[\sigma]^{-1}M$ exists	
R−	$(M \cdot^{\mu} \xi) \mid \sigma = (\uparrow(\xi \mid \sigma))$	if not	
RN	$\cdot \mid \sigma = \cdot$		

Fig. 1. Inversion and Restriction

Lemma 3. *If there exists θ and N such that $\cdot; \Psi \vdash N : A$ and $\Delta; \Gamma \vdash \sigma : \Psi$ and $\Delta; \Gamma \vdash [\![\theta]\!][\sigma]N = [\![\theta]\!]M : [\sigma]A$, then $[\sigma]^{-1}M$ exists and is equal to N, modulo the additional equations generated, if any.*

Proof. Without loss of generality, assume N and M are in η-long β-normal form. We can then proceed by induction on the structure of M.

The case for the new rule **NI** requires showing that if $\Delta; \Gamma \vdash [\![\theta]\!][M \cdot^{\mathsf{i}} \sigma]N = \mathsf{n}$, then $[\sigma]^{-1}\mathsf{n}$ exists and is equal to N. For N under a substitution to be equal to a deBruijn index, it must be itself a deBruijn index, and it can't be 1, since that would not be well-typed: it accesses M from $M \cdot^{\mathsf{i}} \sigma$, violating the implicit contract that a legitimate, relevant term cannot use irrelevant assumptions from the context (respectively, irrelevantly cons-ed terms from a substitution) except under irrelevant application. Therefore there exists a deBruijn index m such that $N = \uparrow\mathsf{m}$. What we know by assumption becomes $\Delta; \Gamma \vdash [\![\theta]\!][M \cdot^{\mathsf{i}} \sigma]\uparrow\mathsf{m} = \mathsf{n}$, and **ShiftCons**-like reasoning again gives $[\sigma]\mathsf{m} = \mathsf{n}$. Now the induction hypothesis applies, giving $\mathsf{m} = [\sigma]^{-1}\mathsf{n}$, which implies $N = \uparrow([\sigma]^{-1}\mathsf{n})$, as required.

For **A**, we have that $[\![\theta]\!][\sigma](N_1 \diamond^{\mathsf{i}} N_2) = [\![\theta]\!](M_1 \diamond^{\mathsf{i}} M_2)$, but we can only apply the induction hypothesis to M_1, N_1, because we have no guarantee of equality under the irrelevant application. However, if M_2 under the inverse of σ does not exist, then **A−** applies, and we still get the required definitional equality.

If $M = u[\xi]$ and $\mathbf{V}-$ applies, then suppose that $P/u, P'/u' \in \theta$. Then we know $[\![\theta]\!][\sigma]N = [[\![\theta]\!]\xi]P$. So the new equation $u[\mathsf{id}] = u'[\xi \,|\, \sigma]$ means $P = [[\![\theta]\!](\xi \,|\, \sigma)]P'$. Thus $[[\![\theta]\!]\xi]P = [[\![\theta]\!]([\xi](\xi \,|\, \sigma))]P' = [\![\theta]\!](u'[[\xi](\xi \,|\, \sigma)])$. By the induction hypothesis, (the new term can be construed as smaller because the substitution applied to the modal variable is now guaranteed to be in the domain of $[\sigma]^{-1}$ by the definition operation) $u'[[\sigma]^{-1}[\xi](\xi \,|\, \sigma)]$ exists and is equal to N, but this is the definition of $[\sigma]^{-1}M$. ■

The restriction operation $\xi \,|\, \sigma$ serves to take ξ and produce a substitution such that $[\xi](\xi \,|\, \sigma)$ is just like ξ only leaving out those terms that are not in the domain of $[\sigma]^{-1}$. The rule $\mathbf{R}+$ in fact needs not pass along even those irrelevant terms that do fall in the domain of $[\sigma]^{-1}$, however. We could have written instead of $\mathbf{R}+$ the two rules

$$\mathbf{RR}+ \qquad (M \cdot^{\mathsf{r}} \xi) \,|\, \sigma = \mathsf{1} \cdot^{\mathsf{r}} (\uparrow(\xi \,|\, \sigma)) \qquad \text{if } [\sigma]^{-1}M \text{ exists}$$

$$\mathbf{RI}+ \qquad (M \cdot^{\mathsf{i}} \xi) \,|\, \sigma = \uparrow(\xi \,|\, \sigma) \qquad \text{if } [\sigma]^{-1}M \text{ exists}$$

without sacrificing correctness. The reason for this is that we never strictly need, during unification, to preserve irrelevantly used terms (i.e. N in $M \diamond^{\mathsf{i}} N$ or M in $M \cdot^{\mathsf{i}} \sigma$) since the equational theory permits them to be replaced by fresh modal variables of the appropriate type. The restriction rules are written as they are, since preserving the terms seems desirable even if not necessary. This issue is treated in some more detail in the next section.

3.3 Unification Transformations

A unification problem is given by

$$\Delta_F; \exists \Delta_B. \bigwedge_{i \in I} \Gamma_i \vdash M_i \stackrel{?}{=} N_i : A_i$$

for modal contexts Δ_F of *free* variables and Δ_B of *bound* variables. These two modal contexts are used to separate the two roles played by modal variables in the unification algorithm. Δ_F is initially populated by those variables for which we seek instantiations, and Δ_B contains variables for which instantiations need to exist for an instantiation of Δ_F to be a unifier. When new variables are added, (for instance, when a flex-flex equation $u[\sigma] = u[\sigma']$ is transformed to $u'[\sigma \cap \sigma']$ to enforce that u is instantiated with some term that behaves the same under σ and σ') they are added to Δ_B, since the instantiations of these new variables is not important to answer the original problem posed.

Therefore formally we say a unification problem is *unified* by a modal substitution $\Delta' \vdash \theta_F : \Delta_F$ if there exists a substitution $\Delta'' \vdash \theta_B : \Delta_B$ for Δ'' extending Δ' such that if

$$\Delta''; [\![\theta_F, \theta_B]\!]\Gamma_i \vdash [\![\theta_F, \theta_B]\!]M_i = [\![\theta_F, \theta_B]\!]N_i : [\![\theta_F, \theta_B]\!]A_i$$

for all i. The notion of most general unifier is defined in the usual way.

Since a unifier is only required to make the terms under substitution equal up to definitional equality, we could at the outset replace every term $M \diamond^{\mathsf{i}} N$ in a problem with $M \diamond^{\mathsf{i}} v[\cdot]$ for a fresh modal variable v of the appropriate type. The fact that we *can* safely effect this kind of replacement is extremely important for solving problems such as

$$\lambda^{\mathsf{r}}_A (c \diamond^{\mathsf{i}} 1) \stackrel{?}{=} \lambda^{\mathsf{r}}_A u[\cdot]$$

No modal substitution for u results in $u[\cdot]$ becoming $c \diamond^{\mathsf{i}} 1$, since the argument to the function is not made available to u, but we can produce a substitution $c \diamond^{\mathsf{i}} v[\cdot]/u$, which produces the equation

$$\lambda^{\mathsf{r}}_A (c \diamond^{\mathsf{i}} 1) \stackrel{?}{=} \lambda^{\mathsf{r}}_A (c \diamond^{\mathsf{i}} v[\cdot])$$

which does hold, by virtue of how equality is defined for $\diamond^{\mathsf{i}}$. Note that with regard to the the problem for finding *closed* substitutions (i.e., containing no modal variables) for all the modal variables, we have merely deferred to the problem of determining whether there indeed exist closed terms at the type A.

So, as stated, it would be possible to replace at the outset all irrelevant arguments with fresh modal variables. This would in no way compromise the correctness of the algorithm, since up to definitional equality, we are changing nothing. Why not do exactly this? It is because we aim to make as easy as possible the problem of finding closed solutions unification is complete, which may, because of the presence of proof irrelevance, require solving (generally undecidable) inhabitation problems. Therefore throughout the algorithm below we take care to preserve, whenever possible, the structure of irrelevant arguments, in the hope that they might later prove useful as witnesses of inhabitation.

The unification algorithm proceeds, as expected, by rewriting equations in a way that maintains the same set of unifiers, towards the goal of a *solved form*

$$\Delta_F; \exists \Delta'_B . \bigwedge_{i \in J} \Psi_j \vdash u_j[\mathsf{id}_{\Psi_j}] \stackrel{?}{=} P_j : A_j$$

(where no modal variables occur in P_i) which witnesses that the mgu of the original system is $P_1/u_1, \ldots, P_m/u_m$. It is very close to the "practical transformations" in the algorithm of Dowek et al. [4] except that we need not cover cases for inverted substitutions applied to yet other substitutions, because the presentation does not use full explicit substitutions. We maintain the invariant that terms in equations are weak head normal, and use the notation $\langle M \rangle$ to denote the weak head normal form of M. We omit several rules which are merely symmetric versions of rules listed, or are straightforward analyses of equations involving constants. We also omit the ordinary contexts Γ, since their role is easily inferred. For instance, **LL** written out fully is $\Gamma \vdash \lambda^{\mu}_A M \stackrel{?}{=} \lambda^{\mu}_A N \to \Gamma \cdot^{\mu} A \vdash \langle M \rangle \stackrel{?}{=} \langle N \rangle$.

The intersection $\sigma \cap \xi$ of two substitutions as in **VV** is defined by

$$\begin{aligned} (\mathsf{n} \cdot^{\mathsf{r}} \sigma) \cap (\mathsf{n} \cdot^{\mathsf{r}} \sigma') &= \mathsf{n} \cdot^{\mathsf{r}} (\sigma \cap \sigma') \\ (\mathsf{n} \cdot^{\mathsf{r}} \sigma) \cap (\mathsf{m} \cdot^{\mathsf{r}} \sigma') &= \sigma \cap \sigma' \qquad \text{if } n \neq m \\ (M \cdot^{\mathsf{i}} \sigma) \cap (M' \cdot^{\mathsf{i}} \sigma') &= M \cdot^{\mathsf{i}} (\sigma \cap \sigma') \end{aligned}$$

AA	$M_1 \diamond^{\mathsf{r}} M_2 \stackrel{?}{=} N_1 \diamond^{\mathsf{r}} N_2 \to M_1 \stackrel{?}{=} N_1 \wedge \langle M_2 \rangle \stackrel{?}{=} \langle N_2 \rangle$		
AAI	$M_1 \diamond^{\mathsf{i}} M_2 \stackrel{?}{=} N_1 \diamond^{\mathsf{i}} N_2 \to M_1 \stackrel{?}{=} N_1$		
LL	$\lambda^\mu_A M \stackrel{?}{=} \lambda^\mu_A N \to \langle M \rangle \stackrel{?}{=} \langle N \rangle$		
TL	$N \stackrel{?}{=} \lambda^\mu_A M \to \langle M \rangle \stackrel{?}{=} \langle (\uparrow N) \diamond^\mu \mathsf{1} \rangle$		
N=	$\mathsf{n} \stackrel{?}{=} \mathsf{n} \to \mathbb{T}$		
N≠	$\mathsf{n} \stackrel{?}{=} \mathsf{m} \to \mathbb{F}$	if $m \neq n$	
NA	$\mathsf{n} \stackrel{?}{=} M_1 \diamond^\mu M_2 \to \mathbb{F}$		
VV	$u[\sigma] \stackrel{?}{=} u[\xi] \to u'[\sigma \cap \xi]$	[1]	
VT	$u[\sigma] \stackrel{?}{=} M \to u[\mathsf{id}] \stackrel{?}{=} [\sigma]^{-1} M$	[2]	
OCI	$u[\sigma] \stackrel{?}{=} M \to u[\sigma] \stackrel{?}{=} [\![v[\mathsf{id}]/u]\!] M$	[3]	
OC	$u[\sigma] \stackrel{?}{=} M \to \mathbb{F}$	[4]	

[1] for u' a fresh variable added to Δ_B
[2] if **VV** does not apply, $[\sigma]^{-1}M$ exists, and u does not occur in M
[3] if **VV** does not apply and u only occurs under the argument side of $\diamond^{\mathsf{i}}$ in M, for v a fresh variable of the same type as u added to Δ_B
[4] if none of **VV**, **VT**, **OCI** applies.

Fig. 2. Unification Transformations

The presence of proof irrelevance in the type system forces several novelties into the algorithm. The rule **AAI** handles comparison of two irrelevant applications. We can neglect trying to solve the equation of the arguments, since we know that $M_1 \diamond^{\mathsf{i}} N_1 = M_2 \diamond^{\mathsf{i}} N_2$ if merely $M_1 = M_2$.

The occurs-check need not (and in fact must not) reject equations $u[\mathsf{id}] = M$ where u occurs in M, if it occurs only "irrelevantly", that is, somewhere inside the arguments of irrelevant applications. For if it does, we can swap in (rule **OCI**) a new modal variable v of the same type, keeping the whole term invariant up to definitional equality.

Rules **VV**, **VT**, and **OCI** use the new definitions of substitution, intersection, and inversion. To verify the correctness of **VV**, we must show that $[\sigma]P = [\xi]P$ if and only if there exists P' such that $P = [\sigma \cap \xi]P'$.[2] But this is established an easy inductive argument. The correctness of the other two rules can be shown using the properties of inversion already proved.

4 Conclusions

With some modifications, an established algorithm for higher-order pattern unification can be made to perform correctly in the presence of proof irrelevance. It

[2] We need not consider the possibility that σ and ξ in the left-hand side of **VV** contain occurrences of u, because if u does occur in them, it only occurs under an irrelevant cons because σ, ξ are pattern, so we can, as above, swap in a fresh modal variable in place of u without loss.

is hoped that future work can do the same for mode, termination, and coverage checking, so that logic programs can be written in a style conscious of which arguments are safe to discard during execution, realizing space and time savings in the distribution and checking of proof terms.

References

1. Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lèvy. Explicit substitutions. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California*, pages 31–46. ACM, 1990.
2. Karl Crary and Susmit Sarkar. A metalogical approach to foundational certified code. Technical Report CMU-CS-03-108, Carnegie Mellon University, January 2003.
3. Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order unification via explicit substitutions. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 366–374, San Diego, California, June 1995. IEEE Computer Society Press.
4. Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 259–273, Bonn, Germany, September 1996. MIT Press.
5. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.
6. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
7. Frank Pfenning. Intensionality, extensionality and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*, 2001.
8. Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1998.
9. Brigitte Pientka and Frank Pfenning. Optimizing higher-order pattern unification. Submitted, 2003.
10. Jason Reed. Higher-order pattern unification and proof irrelevance. Appears in TPHOLs 2002 Track B proceedings, NASA tech report CP-2002-211736, 2002.
11. Jason Reed. Proof irrelevance and strict definitions in a logical framework. Technical Report CMU-CS-02-153, Carnegie Mellon University, 2002.
12. Ekkehard Rohwedder and Frank Pfenning. Mode and termination checking for higher-order logic programs. In Hanne Riis Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1058.