Explicit Substitutions for Contextual Type Theory

Andreas Abel
Theoretical Computer Science, Ludwig-Maximilians-University Munich, Germany
andreas.abel@ifi.lmu.de

Brigitte Pientka
School of Computer Science, McGill University, Montreal, Canada
bpientka@cs.mcgill.ca

In this paper, we present an explicit substitution calculus which distinguishes between ordinary bound variables and meta-variables. Its typing discipline is derived from contextual modal type theory. We first present a dependently typed lambda calculus with explicit substitutions for ordinary variables and explicit meta-substitutions for meta-variables. We then present a weak head normalization procedure which performs both substitutions lazily and in a single pass thereby combining substitution walks for the two different classes of variables. Finally, we describe a bidirectional type checking algorithm which uses weak head normalization and prove soundness.

Keywords: Explicit substitutions, Meta-variables, Logical framework, Contextual modal type theory

1 Introduction

Over the last decade, reasoning and programming with dependent types has received wide attention and several systems provide implementations for dependently typed languages (see for example Agda [BDN09, Nor07], Beluga [PD08, PD10], Delphin [PS08, PS09], Twelf [PS99], etc).

As dependent types become more accepted, it is interesting to better understand how to implement such systems efficiently. While all the systems mentioned support type checking and moreover provide implementations supporting type reconstruction for dependent types, there is a surprising lack in documentation and gap in modelling the theoretical foundations of these implementations. This makes it hard to reproduce some of the ideas, and prevents them from being widely accessible to a broader audience.

A core question in the implementations for dependently typed systems is how to handle substitutions. Let us illustrate the problem in the setting of contextual modal type theory [NPP08], where we not only have ordinary Π-types to abstract over ordinary variables x but also $\Pi^{\square}$-types which allow us to abstract over meta-variables X, and we find the following two elimination rules:

$$\frac{\Delta;\Gamma \vdash M : \Pi x{:}A.B \quad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash M\ N : [N/x]B} \qquad \frac{\Delta;\Gamma \vdash M : \Pi^{\square} X{:}A[\Psi].B \quad \Delta;\Psi \vdash N : A}{\Delta;\Gamma \vdash M\ (\hat{\Psi}.N) : [\![\hat{\Psi}.N/X]\!]B}$$

In the Π-elimination rule, we do not want to apply the substitution N for x in the type B eagerly during type checking, but accumulate all the individual substitutions and apply them simultaneously, if necessary. Similarly, in the $\Pi^{\square}$-elimination rule, we do not want to replace eagerly the meta-variable X with N in the type B but accumulate all meta-substitutions and also apply them simultaneously. In fact, we would like to combine substitution walks for meta-variables and ordinary variables, and simultaneously apply ordinary substitution and meta-substitutions to avoid multiple traversals. This will allow us

Preliminary Report. Final version to appear in:
LFMTP 2010

© A. Abel & B. Pientka
This work is licensed under the
Creative Commons Attribution License.

potentially to detect that two terms are not equal without actually performing a substitution, and in the case of a de Bruijn numbering scheme for variables, we would like to avoid unnecessary renumbering.

Explicit substitutions go back to Abadi et al [ACCL91] and are often central when implementing core algorithms such as type checking or higher-order unification [DHK00]. Many existing implementations of proof assistants such as the Twelf system, Delphin, Beluga, Agda or λProlog use explicit substitutions to combine substitution walks for ordinary variables. A different approach with the same goal of handling substitutions efficiently is the suspension calculus [NW98, LNQ05].

However, meta-variables are often modeled via references thereby avoiding the need to explicitly propagate substitutions for meta-variables. Yet there are multiple reasons why we would like to treat meta-variables non-destructively and be able to handle meta-substitutions explicitly. First, such implementations may be easier to maintain and may be more efficient. Second, in several applications we need to abstract over the remaining free meta-variables in the most general solution found by higher-order unification. For example in type reconstruction we need to store a closed most general type or in tabled higher-order logic programming [Pie03] we want to store explicitly the answer substitution for the meta-variables occurring in a query. Abstraction can be expensive since we need to first traverse a term including the types of all the meta-variables occurring in it and collect all references to meta-variables. Subsequently, we again need to traverse the term including the types of meta-variables and compute their appropriate de Bruijn index. A non-destructive implementation of unification could avoid this explicit abstraction step. To achieve a practical, non-destructive implementation of unification, understanding the interaction of ordinary substitutions with meta-substitutions and handling both lazily is crucial.

While meta-variables are often only introduced internally, i.e., there is no abstraction over meta-variables using a $\Pi^{\square}$-type, some languages such as Beluga have taken the step to distinguish ordinary bound variables and meta-variables already in the source language. Consequently, we find different classes of bound variables, bound ordinary variables and bound meta-variables, and different types, Π- and $\Pi^{\square}$-types. When type-checking Beluga programs, we would like to combine substitution walks for these different classes. Understanding how these two substitutions interact is also crucial for type reconstruction in this setting, since omitted arguments may depend on both kinds of variables.

In this paper, we revisit the ideas of explicit substitutions where we combine substitutions for ordinary variables and meta-variables. In particular, we describe an explicit substitution calculus with first-class meta-variables inspired by contextual modal type theory [NPP08]. We first present a dependently typed lambda calculus with explicit substitutions for ordinary variables and explicit meta-substitutions for meta-variables. We omit here the ability to abstract explicitly over meta-variables which is a straightforward addition and concentrate on the interaction of ordinary substitutions and meta-substitutions. We then present a weak head normalization procedure which performs both substitutions lazily and in a single pass thereby combining substitution walks for the two different classes of variables. Finally, we give an algorithm for definitional equality and present a bidirectional type checking algorithm which employs weak head normalization and show soundness. In the future, we plan to use the presented calculus as a foundation for implementing the Beluga language which supports programming and reasoning with formal systems specified in the logical framework LF.

2 The Calculus: Syntax, Typing, and Equality

Let us first introduce the grammar and typing rules for the dependently typed λ-calculus with meta-variables based on the ideas in [NPP08]. The system we consider is an extension of the logical framework LF with first-class meta-variables. We design the calculus as an extension of previous explicit substitution

calculi such as [ACCL91, DHK00]. These calculi only support ordinary substitutions but not at the same time meta-substitutions.

Our calculus supports general closures on the type and term level. Meta-variables (which sometimes are also called contextual variables) are written as X. Typically, meta-variables occur as a closure $[\sigma]X$, but we will treat this as a special case of the general closure $[\sigma]N$.

To provide a compact representation of the typing rules, we follow the tradition of pure type systems and introduce sorts and expressions where sorts can be either kind or type and expressions include terms, types and kinds. A single syntactic category of expressions helps us avoid duplication in the typing and equality rules for closures $[\sigma]E$ and $[\![\theta]\!]E$. We will write M, A, K, if indeed expressions can only occur as terms M, types A or kinds K.

Sorts	s	$::=$	$\mathsf{kind} \mid \mathsf{type}$	
Expressions	E, F	$::=$	$s \mid a \mid \Pi E.F \mid x_n \mid X_n \mid \lambda\, E \mid F\ E \mid [\sigma]E \mid [\![\theta]\!]E$	
Special cases of expressions:				
Kinds	K	$::=$	$\mathsf{type} \mid \Pi A.K \mid [\sigma]K \mid [\![\theta]\!]K$	
Types	A, B	$::=$	$a \mid A\ M \mid \Pi A.B \mid [\sigma]A \mid [\![\theta]\!]A$	
Terms	M, N	$::=$	$x_n \mid X_n \mid \lambda\, M \mid M\ N \mid [\sigma]N \mid [\![\theta]\!]M$	$(n \geq 1)$
Substitutions	σ, τ	$::=$	$\uparrow^n \mid \sigma, M \mid [\tau]\sigma \mid [\![\theta]\!]\sigma$	$(n \geq 0)$
Meta-substitutions	θ	$::=$	$\Uparrow^n \mid \theta, M \mid [\![\theta]\!]\theta'$	$(n \geq 0)$
Contexts	Γ, Ψ	$::=$	$\cdot \mid \Psi, A$	
Meta-contexts	Δ	$::=$	$\cdot \mid \Delta, \Psi \triangleright A$	

Constants are denoted by letter a, their types/kinds are recorded in a global well-formed signature Σ. We have two different de Bruijn indices x_n and X_n ($n \geq 1$), one for numbering bound variables and one for numbering meta-variables. x_n represents the de Bruijn number n and stands for an ordinary bound variable, while X_n represents the de Bruijn number n but stands for a meta-variable. Due to the two kinds of substitutions, we also have two kinds of closures; the closure of an expression with an ordinary substitution σ and the closure of an expression with a meta-substitution θ. Following the treatment of meta-variables in [NPP08], we describe the type of a meta-variable as $\Psi \triangleright A$ which stands for a meta-variable of type A which may refer to variables in Ψ.

Meta-substitutions provide a term M for a meta-variable X of type $\Psi \triangleright A$. Note that M does not denote a closed term, but a term of type A in the context Ψ and hence may refer to variables from Ψ. In previous presentations where we use names for variables, we hence wrote $\hat{\Psi}.M/X$ to be able to rename the variables in M appropriately. Because bound variables are represented using de Bruijn indices in this paper, we simply write M/X but keep in mind that M is not necessarily closed.

Our calculus also features closures on the level of substitutions and meta-substitutions. For example, we allow the closure $[\sigma]\tau$ which will allow us to lazily treat ordinary substitution composition and the closure $[\![\theta]\!]\sigma$ which will postpone applying θ to the ordinary substitution σ. Similarly, the closure $[\![\theta]\!]\theta'$ for meta-substitutions allows us to lazily compose meta-substitutions. We note the absence of a closure $[\sigma]\theta$. Applying an ordinary substitution σ to a meta-substitution θ simply reduces to θ, since all objects in the meta-substitution are closed objects and cannot be affected by σ. It is hence not meaningful to include a closure $[\sigma]\theta$. We also do not introduce a closure of a context Ψ and a meta-substitution θ. Instead we define $[\![\theta]\!]\Psi$ eagerly by simply pushing the meta-substitution θ to each declaration as

follows: $[\![\theta]\!]\cdot = \cdot$ and $[\![\theta]\!](\Psi, A) = [\![\theta]\!]\Psi, [\![\theta]\!]A$. The length of a context Γ is denoted by $|\Gamma|$ and likewise $|\Delta|$ for meta-contexts.

Expressions

$$\frac{\Delta \vdash \Gamma\ \mathsf{ctx}}{\Delta;\Gamma \vdash \mathsf{type} : \mathsf{kind}} \qquad \frac{\Delta \vdash \Gamma\ \mathsf{ctx} \qquad \Sigma(a) = K}{\Delta;\Gamma \vdash a : K} \qquad \frac{\Delta;\Gamma, A \vdash E : s}{\Delta;\Gamma \vdash \Pi A.E : s}$$

$$\frac{\Delta;\Gamma \vdash A : \mathsf{type}}{\Delta;\Gamma, A \vdash x_1 : [\uparrow^1]A} \qquad \frac{\Delta;\Gamma \vdash x_n : A \quad \Delta;\Gamma \vdash B : \mathsf{type}}{\Delta;\Gamma, B \vdash x_{n+1} : [\uparrow^1]A}$$

$$\frac{\Delta;\Gamma \vdash A : \mathsf{type}}{\Delta, \Gamma \triangleright A; [\Uparrow^1]\Gamma \vdash X_1 : [\Uparrow^1]A} \qquad \frac{\Delta;\Gamma \vdash X_n : A \quad \Delta;\Gamma' \vdash A' : \mathsf{type}}{\Delta, \Gamma' \triangleright A'; [\Uparrow^1]\Gamma \vdash X_{n+1} : [\Uparrow^1]A}$$

$$\frac{\Delta;\Gamma, A \vdash M : B \qquad \Delta;\Gamma, A \vdash B : \mathsf{type}}{\Delta;\Gamma \vdash \lambda M : \Pi A.B} \qquad \frac{\Delta;\Gamma \vdash E : \Pi A.F \quad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash E\ N : [\uparrow^0, N]F}$$

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \quad \Delta;\Psi \vdash E : \mathsf{kind}}{\Delta;\Gamma \vdash [\sigma]E : \mathsf{kind}} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Psi \quad \Delta;\Psi \vdash E : F}{\Delta;\Gamma \vdash [\sigma]E : [\sigma]F}$$

$$\frac{\Delta \vdash \theta : \Delta' \quad \Delta';\Gamma \vdash E : \mathsf{kind}}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]E : \mathsf{kind}} \qquad \frac{\Delta \vdash \theta : \Delta' \quad \Delta';\Gamma \vdash E : F}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]E : [\![\theta]\!]F} \qquad \frac{\Delta;\Gamma \vdash E : F_1 \quad \Delta;\Gamma \vdash F_1 \equiv F_2 : s}{\Delta;\Gamma \vdash E : F_2}$$

Contexts and meta-contexts

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

Ordinary substitutions

$$\frac{\Delta \vdash \Psi, \Gamma\ \mathsf{ctx} \quad |\Gamma| = n}{\Delta;\Psi, \Gamma \vdash \uparrow^n : \Psi} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Psi \quad \Delta;\Psi \vdash A : \mathsf{type} \quad \Delta;\Gamma \vdash M : [\sigma]A}{\Delta;\Gamma \vdash (\sigma, M) : (\Psi, A)}$$

$$\frac{\Delta;\Gamma \vdash \tau : \Psi' \quad \Delta;\Psi' \vdash \sigma : \Psi}{\Delta;\Gamma \vdash [\tau]\sigma : \Psi} \qquad \frac{\Delta \vdash \theta : \Delta' \quad \Delta';\Gamma \vdash \sigma : \Psi}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]\sigma : [\![\theta]\!]\Psi}$$

Meta-substitutions

$$\frac{\vdash \Delta, \Delta'\ \mathsf{mctx} \quad |\Delta'| = n}{\Delta, \Delta' \vdash \Uparrow^n : \Delta} \qquad \frac{\Delta \vdash \theta : \Delta' \quad \Delta';\Gamma \vdash A : \mathsf{type} \quad \Delta;[\![\theta]\!]\Gamma \vdash M : [\![\theta]\!]A}{\Delta \vdash (\theta, M) : \Delta', \Gamma \triangleright A}$$

$$\frac{\Delta \vdash \theta : \Delta_0 \quad \Delta_0 \vdash \theta' : \Delta'}{\Delta \vdash [\![\theta]\!]\theta' : \Delta'}$$

Figure 1: Typing rules for explicit substitution calculus with first-class meta-variables

2.1 Typing rules

In contrast to [HP05], we present the typing rules for LF in pure type system (PTS) style, to avoid rule duplication (which would be substantial for the rules of definitional equality given in the next section).

We will use the following judgments:

	$\vdash$	$\Delta\ \mathsf{mctx}$	Meta-context Δ is well-typed
Δ	$\vdash$	$\Psi\ \mathsf{ctx}$	Context Ψ is well-typed
$\Delta;\Gamma$	$\vdash$	$E : F$	Expression E has "type" F
$\Delta;\Gamma$	$\vdash$	$\sigma : \Psi$	Substitution σ has domain Ψ and range Γ
Δ	$\vdash$	$\theta : \Delta'$	Contextual Substitution θ has domain Δ' and range Δ

The judgement $\Delta;\Gamma \vdash E : F$ subsumes the judgements $\Delta;\Gamma \vdash M : A$ (term M has type A), $\Delta;\Gamma \vdash A : K$ (type family A has kind K) and $\Delta;\Gamma \vdash K : \mathsf{kind}$ (kind K is well-formed).

We present the typing rules in Figure 1 as a type assignment system for expressions E. To improve readability, we use the letters M,N instead of E when we know that we are dealing with a term, and similarly A,B for types and K for kinds.

In the typing rule for λM, the hypothesis $\Delta;\Gamma,A \vdash B : \mathsf{type}$ prevents us to form a λ-abstraction on the type level (for this, we would need $B : \mathsf{kind}$). Lambda on the type level does not increase the expressiveness [Ada05, HP05]. Unlike the system in [HP05], we do not assume that the meta-context Δ and the context Γ are well-formed, but ensure that these are well-formed contexts by adding appropriate typing premises to for example the typing rules for bound variables and meta-variables. We establish separately that contexts are well-formed (see Lemma 1 on page 9) and that the inference rules are valid (see Theorem 4 on page 10).

We concentrate here on explaining the typing rules for bound variables and meta-variables. The typing rules for bound variables essentially peel off one type declaration in the context Γ until we encounter the variable x_1. The typing premises guarantee that the meta-context Δ and the context Γ and the type A of the bound variable all are well-typed. The rule for meta-variables are built in a similar fashion as the typing rules for bound variables peeling off type declarations from the meta-context Δ until we encounter the meta-variable X_1.

2.2 Definitional Equality

In this section, we describe a typed $\beta\eta$-equality judgement on expressions, ordinary substitutions, and meta-substitutions. We will use the following judgments:

$\Delta;\Gamma \vdash E_1 \equiv E_2 : F$	Expressions E_1 and E_2 are equal at "type" F
$\Delta;\Gamma \vdash \sigma_1 \equiv \sigma_2 : \Psi$	Substitutions σ_1 and σ_2 are equal at domain Ψ
$\Delta \vdash \theta_1 \equiv \theta_2 : \Delta'$	Meta-substitutions θ_1 and θ_2 are equal at domain Δ'

The judgement $\Delta;\Gamma \vdash E_1 \equiv E_2 : F$ subsumes the judgements $\Delta;\Gamma \vdash K_1 \equiv K_2 : \mathsf{kind}$ (kinds K_1 and K_2 are equal), $\Delta;\Gamma \vdash A_1 \equiv A_2 : K$ (types A_1 and A_2 are equal of kind K) and $\Delta;\Gamma \vdash M_1 \equiv M_2 : A$ (terms M_1 and M_2 are equal of type A).

These judgements are all congruences, i. e., we have equivalence rules (reflexivity, symmetry, transitivity) and a congruence rule for each syntactic construction. For instance, this is one of congruence rule for substitutions and the type conversion rule:

$$\frac{\Delta;\Gamma \vdash M \equiv M' : [\sigma]A \quad \Delta;\Gamma \vdash A : \mathsf{type} \quad \Delta;\Gamma \vdash \sigma \equiv \sigma' : \Psi}{\Delta;\Gamma \vdash (\sigma, M) \equiv (\sigma', M') : \Psi, A} \qquad \frac{\Delta;\Gamma \vdash E \equiv E' : F \quad \Delta;\Gamma \vdash F \equiv F' : s}{\Delta;\Gamma \vdash E \equiv E' : F'}$$

The remaining rules for definitional equality fall into two classes: the computational laws for ordinary substitutions (Figure 2) and the computational laws for meta-substitutions (Figure 3). Both sets of rules

follow the same principle. They are grouped into identity and composition rules, propagation and reduction rules. For ordinary substitutions we also include β-reduction. For meta-substitutions, there is no equivalent β-reduction rule since we do not support abstraction over meta-variables. However, we add propagation into ordinary substitutions. Note that pushing a meta-substitution inside a lambda-abstraction or a Π-type does not require a shift of the indices, since indices of ordinary bound variables are distinct from indices of meta-variables and no capture can occur. There is no reduction for $[\sigma][\![\theta]\!]M$: an ordinary substitution cannot in general be pushed past a meta-substitution, it has to wait for the meta-substitution to be resolved.

To illustrate the definitional equality rules, we show how to derive $\Delta;\Gamma \vdash [\sigma,M]x_{n+1} \equiv [\sigma]x_n : [\sigma]A$ which also demonstrates that such a rule is admissible. Transitivity is essential to assemble the following sub-derivations.

Step 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta;\Psi \vdash x_n : A}{\Delta;\Psi,B \vdash x_{n+1} : [\uparrow^1]A}\,\text{Weakening}}{\Delta;\Psi,B \vdash x_{n+1} \equiv [\uparrow^1]x_n : [\uparrow^1]A}\,\text{Reduction}}{\Delta;\Gamma \vdash [\sigma,M]x_{n+1} \equiv [\sigma,M][\uparrow^1]x_n : [\sigma,M][\uparrow^1]A}\,\text{Congruence}}{\Delta;\Gamma \vdash [\sigma,M]x_{n+1} \equiv [\sigma,M][\uparrow^1]x_n : [[\sigma,M]\uparrow^1]A}\,\text{Composition, Conversion}$$

Step 2:

$$\dfrac{\dfrac{\Delta;\Gamma \vdash \sigma,M : \Psi,B \quad \Delta;\Psi,B \vdash \uparrow^1 : \Psi \quad \Delta;\Psi \vdash x_n : A}{\Delta;\Gamma \vdash [\sigma,M][\uparrow^1]x_n \equiv [[\sigma,M]\uparrow^1]x_n : [[\sigma,M]\uparrow^1]A}\,\text{Composition} \quad \dfrac{\dfrac{\mathscr{D}}{\Delta;\Gamma \vdash [\sigma,M]\uparrow^1 \equiv \sigma : \Psi}}{\Delta;\Gamma \vdash [[\sigma,M]\uparrow^1]A \equiv [\sigma]A : \mathsf{type}}\,\text{Congruence}}{\Delta;\Gamma \vdash [\sigma,M][\uparrow^1]x_n \equiv [[\sigma,M]\uparrow^1]x_n : [\sigma]A}\,\text{Conversion}$$

Step 3:

$$\dfrac{\dfrac{\mathscr{D}}{\Delta;\Gamma \vdash [\sigma,M]\uparrow^1 \equiv \sigma : \Psi}}{\Delta;\Gamma \vdash [[\sigma,M]\uparrow^1]x_n \equiv [\sigma]x_n : [\sigma]A}\,\text{Congruence}$$

where

$$\mathscr{D} = \dfrac{\dfrac{\Delta;\Gamma \vdash \sigma,M : \Psi,B}{\Delta;\Gamma \vdash [\sigma,M]\uparrow^1 = [\sigma]\uparrow^0 : \Psi}\,\text{Pairing} \quad \dfrac{\Delta;\Gamma \vdash \sigma : \Psi}{\Delta;\Gamma \vdash [\sigma]\uparrow^0 \equiv \sigma : \Psi}\,\text{Category Laws}}{\Delta;\Gamma \vdash [\sigma,M]\uparrow^1 \equiv \sigma : \Psi}\,\text{Transitivity}$$

Similarly, we can show that $\Delta;\Gamma \vdash [\![\theta,M]\!]X_{n+1} \equiv [\![\theta]\!]X_n : [\![\theta]\!]A$ is admissible.

2.2.1 Extensionality Laws

As mentioned earlier, we take into account β-reductions and η-expansions. In particular, we consider η-rules for ordinary substitutions as well as meta-substitutions.

$$\dfrac{\Delta;\Gamma \vdash M : \Pi A.B}{\Delta;\Gamma \vdash M \equiv \lambda(([\uparrow^1]M)\, x_1) : \Pi A.B}$$

$$\dfrac{\Delta \vdash \Gamma,A,\Gamma'\ \mathsf{ctx} \qquad |\Gamma'| = n}{\Delta;\Gamma,A,\Gamma' \vdash \uparrow^n \equiv (\uparrow^{n+1}, x_{n+1}) : \Gamma,A} \qquad \dfrac{\vdash \Delta,\Gamma \triangleright A,\Delta'\ \mathsf{mctx} \qquad |\Delta'| = n}{\Delta,\Gamma \triangleright A,\Delta' \vdash \Uparrow^n \equiv (\Uparrow^{n+1}, X_{n+1}) : \Delta,\Gamma \triangleright A}$$

β-Reduction

$$\frac{\Delta;\Gamma,A \vdash M : B \qquad \Delta;\Gamma,A \vdash B : \mathsf{type} \qquad \Delta;\Gamma \vdash N : A}{\Delta;\Gamma \vdash (\lambda M)N \equiv [\uparrow^0, N]M : [\uparrow^0, N]B}$$

Substitution Propagation: Identity and Composition

$$\frac{\Delta;\Gamma \vdash E : F}{\Delta;\Gamma \vdash [\uparrow^0]E \equiv E : F} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Gamma' \qquad \Delta;\Gamma' \vdash \tau : \Psi \qquad \Delta;\Psi \vdash E : F}{\Delta;\Gamma \vdash [\sigma][\tau]E \equiv [[\sigma]\tau]E : [[\sigma]\tau]F}$$

Substitution Propagation: Constants

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi}{\Delta;\Gamma \vdash [\sigma]\mathsf{type} \equiv \mathsf{type} : \mathsf{kind}} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Psi \qquad \Delta;\Gamma \vdash a : K}{\Delta;\Gamma \vdash [\sigma]a \equiv a : [\sigma]K}$$

Substitution Propagation: Variable Lookup

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \qquad \Delta;\Psi \vdash A : \mathsf{type} \qquad \Delta;\Gamma \vdash M : [\sigma]A}{\Delta;\Gamma \vdash [\sigma, M]x_1 \equiv M : [\sigma]A} \qquad \frac{\Delta;\Gamma \vdash x_{n+1} : A}{\Delta;\Gamma \vdash x_{n+1} \equiv [\uparrow^1]x_n : A}$$

Substitution Propagation: Pushing into Expression Constructions

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \qquad \Delta;\Psi,A \vdash F : s}{\Delta;\Gamma \vdash [\sigma](\Pi A.F) \equiv \Pi\,[\sigma]A.\,[[\uparrow^1]\sigma, x_1]F : s}$$

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \qquad \Delta;\Psi,A \vdash M : B \qquad \Delta;\Psi,A \vdash B : \mathsf{type}}{\Delta;\Gamma \vdash [\sigma](\lambda M) \equiv \lambda\,[[\uparrow^1]\sigma, x_1]M : \Pi\,[\sigma]A.\,[[\uparrow^1]\sigma, x_1]B}$$

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \qquad \Delta;\Psi \vdash E : \Pi A.F \qquad \Delta;\Psi \vdash N : A}{\Delta;\Gamma \vdash [\sigma](E\,N) \equiv [\sigma]E\;[\sigma]N : [\sigma, [\sigma]N]F}$$

Substitution Reductions: Pairing and Shifting

$$\frac{\Delta;\Gamma \vdash (\sigma, M) : \Psi, \Psi', A \quad |\Psi'| = n}{\Delta;\Gamma \vdash [\sigma, M]\uparrow^{n+1} \equiv [\sigma]\uparrow^n : \Psi} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Psi' \qquad \Delta;\Psi' \vdash (\tau, M) : \Psi, A}{\Delta;\Gamma \vdash [\sigma](\tau, M) \equiv ([\sigma]\tau, [\sigma]M) : \Psi, A}$$

$$\frac{\Delta \vdash \Gamma, \Gamma_1, \Gamma_2\ \mathsf{ctx} \quad |\Gamma_1| = m \quad |\Gamma_2| = n}{\Delta;\Gamma, \Gamma_1, \Gamma_2 \vdash [\uparrow^n]\uparrow^m \equiv \uparrow^{n+m} : \Gamma}$$

Substitution Reductions: Category Laws

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi}{\Delta;\Gamma \vdash [\uparrow^0]\sigma \equiv \sigma : \Psi} \qquad \frac{\Delta;\Gamma \vdash \sigma : \Psi}{\Delta;\Gamma \vdash [\sigma]\uparrow^0 \equiv \sigma : \Psi}$$

$$\frac{\Delta;\Gamma_1 \vdash \sigma_1 : \Gamma_2 \qquad \Delta;\Gamma_2 \vdash \sigma_2 : \Gamma_3 \qquad \Delta;\Gamma_3 \vdash \sigma_3 : \Gamma_4}{\Delta;\Gamma_1 \vdash [\sigma_1][\sigma_2]\sigma_3 \equiv [[\sigma_1]\sigma_2]\sigma_3 : \Gamma_4}$$

Figure 2: Computational Laws I: β and substitutions

Meta-Substitution Propagation: Identity and Composition

$$\frac{\Delta;\Gamma \vdash E : F}{\Delta;\Gamma \vdash [\![\Uparrow^0]\!]E \equiv E : F} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta' \vdash \theta' : \Delta'' \qquad \Delta'';\Gamma \vdash E : F}{\Delta;[\![[\![\theta]\!]\theta']\!]\Gamma \vdash [\![\theta]\!][\![\theta']\!]E \equiv [\![[\![\theta]\!]\theta']\!]E : [\![[\![\theta]\!]\theta']\!]F}$$

Meta-Substitution Propagation: Constants and Ordinary Variables

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta' \vdash \Gamma\ \mathsf{ctx}}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]\mathsf{type} \equiv \mathsf{type} : \mathsf{kind}} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash a : K}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]a \equiv a : [\![\theta]\!]K} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash x_n : A}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!]x_n \equiv x_n : [\![\theta]\!]A}$$

Meta-Substitution Propagation: Meta-variable Lookup

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash A : \mathsf{type} \qquad \Delta;[\![\theta]\!]\Gamma \vdash M : [\![\theta]\!]A}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta,M]\!]X_1 \equiv M : [\![\theta]\!]A} \qquad \frac{\Delta;\Gamma \vdash X_{n+1} : A}{\Delta;\Gamma \vdash X_{n+1} \equiv [\![\Uparrow^1]\!]X_n : A}$$

Meta-Substitution Propagation: Pushing into Expression Constructions

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma,A \vdash F : s}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!](\Pi A.F) \equiv \Pi\,[\![\theta]\!]A.\,[\![\theta]\!]F : s} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma,A \vdash M : B \qquad \Delta';\Gamma,A \vdash B : \mathsf{type}}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!](\lambda M) \equiv \lambda\,[\![\theta]\!]M : \Pi\,[\![\theta]\!]A.\,[\![\theta]\!]B}$$

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash E : \Pi A.F \qquad \Delta';\Gamma \vdash N : A}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!](E\,N) \equiv [\![\theta]\!]E\ [\![\theta]\!]N : [\uparrow^1,[\![\theta]\!]N]F} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash \sigma : \Psi \qquad \Delta';\Psi \vdash E : F}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!][\sigma]E \equiv [[\![\theta]\!]\sigma][\![\theta]\!]E : [[\![\theta]\!]\sigma][\![\theta]\!]F}$$

Meta-Substitution Propagation: Pushing into Ordinary Substitutions

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta' \vdash \Gamma,\Gamma'\ \mathsf{ctx} \qquad |\Gamma'| = n}{\Delta;[\![\theta]\!]\Gamma,[\![\theta]\!]\Gamma' \vdash [\![\theta]\!]\uparrow^n \equiv \uparrow^n : [\![\theta]\!]\Gamma}$$

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash \sigma : \Psi \qquad \Delta';\Psi \vdash A : \mathsf{type} \qquad \Delta';\Gamma \vdash M : [\sigma]A}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!](\sigma,M) \equiv ([\![\theta]\!]\sigma,[\![\theta]\!]M) : [\![\theta]\!]\Psi,[\![\theta]\!]A}$$

$$\frac{\Delta \vdash \theta : \Delta' \qquad \Delta';\Gamma \vdash \tau : \Psi' \qquad \Delta';\Psi' \vdash \sigma : \Psi}{\Delta;[\![\theta]\!]\Gamma \vdash [\![\theta]\!][\tau]\sigma \equiv [[\![\theta]\!]\tau][\![\theta]\!]\sigma : [\![\theta]\!]\Psi} \qquad \frac{\Delta \vdash \theta : \Delta' \qquad \Delta' \vdash \theta' : \Delta'' \qquad \Delta'';\Gamma \vdash \sigma : \Psi}{\Delta;[\![[\![\theta]\!]\theta']\!]\Gamma \vdash [\![\theta]\!][\![\theta']\!]\sigma \equiv [\![[\![\theta]\!]\theta']\!]\sigma : [\![[\![\theta]\!]\theta']\!]\Psi}$$

Meta-Substitution Reductions: Pairing and Shifting

$$\frac{\Delta \vdash (\theta,M) : \Delta_0,\Delta_0',\Gamma \triangleright A \quad |\Delta_0'| = n}{\Delta \vdash [\![\theta,M]\!]\Uparrow^{n+1} \equiv [\![\theta]\!]\Uparrow^n : \Delta_0} \qquad \frac{\Delta \vdash \theta : \Delta_0' \qquad \Delta_0' \vdash (\theta',M) : \Delta_0,\Gamma \triangleright A}{\Delta \vdash [\![\theta]\!](\theta',M) \equiv ([\![\theta]\!]\theta',[\![\theta]\!]M) : \Delta_0,\Gamma \triangleright A}$$

$$\frac{\vdash \Delta,\Delta_1,\Delta_2\ \mathsf{mctx} \quad |\Delta_1| = m \quad |\Delta_2| = n}{\Delta,\Delta_1,\Delta_2 \vdash [\![\Uparrow^n]\!]\Uparrow^m \equiv \Uparrow^{n+m} : \Delta}$$

Meta-Substitution Reductions: Category Laws

$$\frac{\Delta \vdash \theta : \Delta_0}{\Delta \vdash [\![\Uparrow^0]\!]\theta \equiv \theta : \Delta_0} \qquad \frac{\Delta \vdash \theta : \Delta_0}{\Delta \vdash [\![\theta]\!]\Uparrow^0 \equiv \theta : \Delta_0} \qquad \frac{\Delta_1 \vdash \theta_1 : \Delta_2 \qquad \Delta_2 \vdash \theta_2 : \Delta_3 \qquad \Delta_3 \vdash \theta_3 : \Delta_4}{\Delta_1 \vdash [\![\theta_1]\!][\![\theta_2]\!]\theta_3 \equiv [\![[\![\theta_1]\!]\theta_2]\!]\theta_3 : \Delta_4}$$

Figure 3: Computational Laws II: Meta-substitution

2.3 Properties

Next, we prove some standard properties about the presented type assignment system. First, we show that contexts are indeed well-formed.

Lemma 1 (Context well-formedness)

1. *If* $\Delta, \Delta' \vdash J$ *or* $\Delta, \Delta'; \Gamma \vdash J$ *then* $\vdash \Delta$ mctx.
2. *If* $\Delta \vdash \theta : \Delta'$ *or* $\Delta \vdash \theta \equiv \theta' : \Delta'$ *then* $\vdash \Delta'$ mctx.
3. *If* $\Delta; \Gamma, \Gamma' \vdash J$ *then* $\Delta \vdash \Gamma$ ctx.
4. *If* $\Delta; \Gamma \vdash \sigma : \Psi$ *or* $\Delta; \Gamma \vdash \sigma \equiv \sigma' : \Psi$ *then* $\Delta \vdash \Psi$ ctx.

The height of the output derivation is bounded by the height of the input derivation, in all cases.

Proof. By simultaneous induction over all judgments. □

The following inversion theorem for typing is standard for PTSs and are necessary due to the type conversion rule which makes inversion a non-obvious property. It allows us to classify expressions into terms, types, kinds, and the sort kind. We write $\Delta; \Gamma \vdash E \equiv E'$ if there exists a sort s such that $\Delta; \Gamma \vdash E \equiv E' : s$.

Theorem 2 (Inversion of typing)

1. *There is no derivation of* $\Delta; \Gamma \vdash \mathsf{kind} : E$.
2. *If* $\Delta; \Gamma \vdash \mathsf{type} : E$ *then* $E = \mathsf{kind}$.
3. *If* $\Delta; \Gamma \vdash a : E$ *then* $\Delta; \Gamma \vdash E \equiv \Sigma(a)$.
4. *If* $\Delta; \Gamma \vdash \Pi A.E : F$ *then* $\Delta; \Gamma \vdash A : \mathsf{type}$ *and* $\Delta; \Gamma, A \vdash E : F$ *and either* $F = \mathsf{kind}$ *or* $\Delta; \Gamma \vdash F \equiv \mathsf{type}$.
5. *If* $\Delta; \Gamma \vdash x_{n+1} : A$ *then* $\Gamma = \Gamma_1, A', \Gamma_2$ *with* $|\Gamma_2| = n$ *and* $\Delta; \Gamma \vdash A \equiv [\uparrow^{n+1}]A'$.
6. *If* $\Delta; \Gamma \vdash X_{n+1} : A$ *then* $\Delta = \Delta_1, \Gamma' \triangleright A', \Delta_2$ *with* $|\Delta_2| = n$ *and* $\Gamma = [\![\Uparrow^{n+1}]\!]\Gamma'$ *and* $\Delta; \Gamma \vdash A \equiv [\![\Uparrow^{n+1}]\!]A'$.
7. *If* $\Delta; \Gamma \vdash \lambda M : C$ *then there are* A, B *such that* $\Delta; \Gamma \vdash C \equiv \Pi A.B$ *and* $\Delta; \Gamma \vdash A : \mathsf{type}$ *and* $\Delta; \Gamma, A \vdash B : \mathsf{type}$ *and* $\Delta; \Gamma, A \vdash M : B$.
8. *If* $\Delta; \Gamma \vdash E\,N : C$ *then there are* A, F *such that* $\Delta; \Gamma \vdash E : \Pi A.F$ *and* $\Delta; \Gamma \vdash N : A$ *and* $\Delta; \Gamma \vdash C \equiv [\uparrow^0, N]F$.
9. *If* $\Delta; \Gamma \vdash [\sigma]E : F$ *then there are* Ψ, F' *such that* $\Delta; \Gamma \vdash \sigma : \Psi$ *and* $\Delta; \Psi \vdash E : F'$ *and* $\Delta; \Gamma \vdash E \equiv [\sigma]F'$.
10. *If* $\Delta; \Gamma \vdash [\![\theta]\!]E : F$ *then there are* Δ', Γ', F' *such that* $\Delta \vdash \theta : \Delta'$ *and* $\Delta; \Gamma' \vdash E : F'$ *and* $\Gamma = [\![\theta]\!]\Gamma'$ *and* $\Delta; \Gamma \vdash F \equiv [\![\theta]\!]F'$.

Proof. By induction on the typing derivation, peeling off the type conversion steps and combining them with transitivity. □

Expression E is a *kind* if $\Delta; \Gamma \vdash E : \mathsf{kind}$ for some Δ, Γ, it is a *type family* if $\Delta; \Gamma \vdash E : K$ for some kind K and some Δ, Γ, and it is a *term* if $\Delta; \Gamma \vdash E : A$ for some A, Δ, Γ with $\Delta; \Gamma \vdash A : \mathsf{type}$.

The following inversion statement for meta-variables under a substitution is crucial for the correctness of algorithmic equality (Sec. 3.2) and bidirectional type checking (Sec. 4).

Corollary 3 *If* $\Delta; \Gamma \vdash [\sigma]X_m : A$ *then* $\Delta = \Delta_1, \Psi \triangleright A', \Delta_2$ *with* $|\Delta_2| = m - 1$ *and* $\Delta; \Gamma \vdash \sigma : [\![\Uparrow^m]\!]\Psi$ *and* $\Delta; \Gamma \vdash A \equiv [\sigma][\![\Uparrow^m]\!]A'$.

Theorem 4 (Syntactic Validity)

1. *If* $\Delta;\Gamma \vdash E : F$ *or* $\Delta;\Gamma \vdash E_1 \equiv E_2 : F$ *then* $\Delta;\Gamma \vdash F : s$ *for some sort.*
2. *If* $\Delta;\Gamma \vdash E \equiv E' : F$ *then* $\Delta;\Gamma \vdash E : F$ *and* $\Delta;\Gamma \vdash E' : F$.
3. *If* $\Delta \vdash \theta \equiv \theta' : \Delta'$ *then* $\Delta \vdash \theta : \Delta'$ *and* $\Delta \vdash \theta' : \Delta'$.
4. *If* $\Delta;\Gamma \vdash \sigma \equiv \sigma' : \Psi$ *then* $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Gamma \vdash \sigma' : \Psi$.

Proof. By simultaneous induction over all judgments. □

3 Evaluation and Algorithmic Equality

In this section, we define a weak head normalization strategy together with algorithmic equality. The goal is to treat ordinary substitutions and meta-substitutions lazily; in particular, we aim to postpone shifting of substitutions until necessary. For the treatment of LF, an untyped algorithmic equality is sufficient. The design of the algorithm follows Coquand [Coq91] with refinements from joint work with the first author [AC07]. In this article, we only show soundness of the algorithm; completeness can be proven using techniques of the cited works. However, an adaptation to de Bruijn style and explicit substitutions is necessary; we leave the details to future work.

We first characterize our normal forms by defining normal and neutral expressions where expressions include terms, types, and kinds. Normal forms are exactly the expressions we can type-check with a bidirectional algorithm (see Section 4). Note that type checking normal forms is sufficient in practice, since the input to the type checker, written by a user, is almost always in β-normal form (or can be turned into normal form by introducing typed let-definitions).

Normal substitutions are built out of normal expressions. However, it is worth keeping in mind that our typing rules will ensure that they only contain terms and not types, since we do not support type-level variables. Our normal forms are only β-normal, not necessarily η-long. Only meta-variables are associated with an ordinary normal substitution, all other closures have been eliminated.

$$\begin{array}{lllll} \text{Normal expressions} & V & ::= & s \mid \Pi V.V' \mid \lambda V \mid U \\ \text{Neutral expressions} & U & ::= & a \mid x_n \mid [\nu]X_n \mid U\ V \end{array}$$

$$\begin{array}{lllll} \text{Normal substitutions} & \nu & ::= & \uparrow^n \mid (\nu, V) \end{array}$$

Next, we define weak head normal forms (whnf). Since we want to treat ordinary substitutions and meta-substitutions lazily and in particular want to postpone the complete computation of their compositions, we cannot require that substitutions and meta-substitutions are already in normal form. Hence, we introduce environments ρ for ordinary substitutions and similarly meta-substitutions η for describing substitutions and meta-substitutions that are in weak head normal form. Closures are expressions E in an environment ρ and a meta-environment η. It is convenient to also treat variables x_n as closures. These arise when stepping under a binder in type and equality checking and are the synonym of Coquand's generic values [Coq96].

$$\begin{array}{lllll} \text{Weak head normal forms} & W & ::= & \mathsf{type} \mid [\rho][\![\eta]\!]\Pi A.B \mid [\rho][\![\eta]\!]\lambda M \mid H \\ \text{Neutral weak head normal forms} & H & ::= & a \mid x_n \mid [\rho]X_n \mid H\ L \\ \text{Closures} & L & ::= & x_n \mid [\rho][\![\eta]\!]E \\ \\ \text{Environments} & \rho & ::= & \uparrow^n \mid (\rho, L) \mid [\uparrow^n]\rho \\ \text{Meta-environments} & \eta & ::= & \Uparrow^n \mid (\eta, M) \end{array}$$

Our weak head normal forms and closures combine substitutions and meta-substitutions and our whnf-reduction strategy simultaneously treats substitutions and meta-substitutions. Instead of coupling expressions with two suspended substitutions, we could have introduced a joint simultaneous substitutions and closures built with them. The path taken in this paper builds on the individual substitution operations instead of defining a new joint substitution operation. To clarify the nature and the interplay of ordinary substitutions and meta-substitutions it is helpful to consider the typing rule of closures $[\rho][\![\eta]\!]E$:

$$\frac{\Delta;\Psi \vdash \rho : [\![\eta]\!]\Psi' \quad \Delta \vdash \eta : \Delta' \quad \Delta';\Psi' \vdash E : F}{\Delta;\Psi \vdash [\rho][\![\eta]\!]E : [\rho][\![\eta]\!]F}$$

Intuitively, this means to obtain an expression E' which makes actually sense in Δ and Ψ, we first compute $[\![\eta]\!]E$ and subsequently apply the ordinary substitution ρ to arrive at $E' \equiv ([\rho][\![\eta]\!]E)$.

Shift propagation. While we treat shifts in the environment as an explicit operation—to avoid a traversal when lifting an environment under a binder—, shifting a closure or a neutral weak head normal form can be implemented inexpensively. Let shifting $\mathsf{shift}^n\, L$ of a closure L be defined by $\mathsf{shift}^n\, x_m = x_{n+m}$ and $\mathsf{shift}^n\,([\rho][\![\eta]\!]E) = [[\uparrow^n]\rho][\![\eta]\!]E$. It is extended to shifting of neutral weak head normal forms H by $\mathsf{shift}^n\,(H\ L) = (\mathsf{shift}^n\, H)\ (\mathsf{shift}^n\, L)$ and $\mathsf{shift}^n\,([\rho]X_m) = [[\uparrow^n]\rho]X_m$ and $\mathsf{shift}^n\, a = a$.

3.1 Weak head evaluation

Our weak head evaluation strategy will postpone propagation of substitutions into an expression until necessary. Treating substitutions lazily seems to be beneficial as also supported by the experimental analysis on lazy vs eager reduction strategies for substitutions by Nadathur and his collaborators [LNQ05]. We present the algorithm for weak head normalization in Figure 4. We define a function $\mathsf{whnf}\ L$ where L is either a variable x_n or a proper closure $[\rho][\![\eta]\!]E$. The function whnf is then defined recursively on E.

To support the lazy evaluation of substitutions, our weak head normalization algorithm relies on the definition of two functions, namely $\mathsf{Env}\ \eta\ \theta$ and $\mathsf{env}\ \rho\ \eta\ \sigma$. Both functions are defined recursively over the last argument, i.e., Env is inductively defined over θ and env is inductively defined over σ. When we encounter a closure of $[\sigma]\tau$ (or $[\![\theta]\!]\theta'$ resp.), we compute first the environment corresponding to σ and subsequently we compute the environment for τ. This strategy allows us to avoid unnecessary shifting of de Bruijn indices.

In addition, whnf relies on a lookup function to retrieve the i-th element of a substitution which corresponds to the index i. Such lookup functions are defined for both, ordinary variables and meta-variables.

Next, we prove that types are preserved when computing weak head normal forms and that the computation is sound with regard to the specification of definitional equality. Note that at this point termination is only clear for the lookup and substitution evaluation functions. For whnf and evaluating application @, soundness is conditional on termination.

Theorem 5 (Subject reduction) *Let* $\Delta \vdash \eta : \Delta'$.

1. *If* $\Delta' \vdash \theta : \Delta''$ *then* $\Delta \vdash \mathsf{Env}\ \eta\ \theta \equiv [\![\eta]\!]\theta : \Delta''$.
2. *If* $\Delta';\Psi \vdash \sigma : \Psi'$ *and* $\Delta;\Gamma \vdash \rho : [\![\eta]\!]\Psi$ *then* $\Delta;\Gamma \vdash \mathsf{env}\ \rho\ \eta\ \sigma \equiv [\rho][\![\eta]\!]\sigma : [\![\eta]\!]\Psi'$.
3. *If* $\Delta';\Psi \vdash X_m : A$ *then* $\Delta;[\![\eta]\!]\Psi \vdash \mathsf{Lookup}\ \eta\ X_m \equiv [\![\eta]\!]X_m : [\![\eta]\!]A$.
4. *If* $\Delta;\Psi \vdash x_m : A$ *and* $\Delta;\Gamma \vdash \rho : \Psi$ *then* $\Delta;\Gamma \vdash \mathsf{lookup}\ \rho\ x_m \equiv [\rho]x_m : [\rho]A$.

Meta-substitution evaluation $\mathsf{Env}\ \eta\ \theta$ computes the meta-environment form of $[\![\eta]\!]\theta$.

$$
\begin{array}{lcl}
\mathsf{Env}\ \Uparrow^m\ \Uparrow^n & = & \Uparrow^{m+n} \\
\mathsf{Env}\ (\eta, M)\ \Uparrow^{n+1} & = & \mathsf{Env}\ \eta\ \Uparrow^n \\
\mathsf{Env}\ \eta\ (\theta, M) & = & (\mathsf{Env}\ \eta\ \theta, [\![\eta]\!]M) \\
\mathsf{Env}\ \eta\ [\![\theta]\!]\theta' & = & \mathsf{Env}\ (\mathsf{Env}\ \eta\ \theta)\ \theta'
\end{array}
$$

Substitution evaluation $\mathsf{env}\ \rho\ \eta\ \sigma$ computes the environment form of $[\rho][\![\eta]\!]\sigma$.

$$
\begin{array}{lcl}
\mathsf{env}\ ([\uparrow^k]\rho)\ \eta\ \sigma & = & [\uparrow^k](\mathsf{env}\ \rho\ \eta\ \sigma) \\
\mathsf{env}\ \rho\ \eta\ \uparrow^0 & = & \rho \\
\mathsf{env}\ \uparrow^k\ \eta\ \uparrow^n & = & \uparrow^{k+n} \\
\mathsf{env}\ (\rho, L)\ \eta\ \uparrow^{n+1} & = & \mathsf{env}\ \rho\ \eta\ \uparrow^n \\
\mathsf{env}\ \rho\ \eta\ (\sigma, M) & = & (\mathsf{env}\ \rho\ \eta\ \sigma, [\rho][\![\eta]\!]M) \\
\mathsf{env}\ \rho\ \eta\ ([\sigma]\tau) & = & \mathsf{env}\ (\mathsf{env}\ \rho\ \eta\ \sigma)\ \eta\ \tau \\
\mathsf{env}\ \rho\ \eta\ ([\![\theta]\!]\sigma) & = & \mathsf{env}\ \rho\ (\mathsf{Env}\ \eta\ \theta)\ \sigma
\end{array}
$$

Meta-variable lookup $\mathsf{Lookup}\ \eta\ X_m$ retrieves the binding of X_m in meta-environment η.

$$
\begin{array}{lcl}
\mathsf{Lookup}\ \Uparrow^n\ X_m & = & X_{n+m} \\
\mathsf{Lookup}\ (\eta, E)\ X_1 & = & E \\
\mathsf{Lookup}\ (\eta, E)\ X_{m+1} & = & \mathsf{Lookup}\ \eta\ X_m
\end{array}
$$

Variable lookup $\mathsf{lookup}\ \rho\ x_m$ computes the closure form of $[\rho]x_m$.

$$
\begin{array}{lcl}
\mathsf{lookup}\ \uparrow^n\ x_m & = & x_{n+m} \\
\mathsf{lookup}\ (\rho, L)\ x_1 & = & L \\
\mathsf{lookup}\ (\rho, L)\ x_{m+1} & = & \mathsf{lookup}\ \rho\ x_m \\
\mathsf{lookup}\ ([\uparrow^n]\rho)\ x_m & = & \mathsf{shift}^n\,(\mathsf{lookup}\ \rho\ x_m)
\end{array}
$$

Weak head evaluation $\mathsf{whnf}\ L$ computes the weak head normal form of closure L.

$$
\begin{array}{lcl}
\mathsf{whnf}\ x_m & = & x_m \\
\mathsf{whnf}\ [\rho][\![\eta]\!]s & = & s \\
\mathsf{whnf}\ [\rho][\![\eta]\!]a & = & a \\
\mathsf{whnf}\ [\rho][\![\eta]\!]x_m & = & \mathsf{whnf}\ (\mathsf{lookup}\ \rho\ x_m) \\
\mathsf{whnf}\ [\rho][\![\Uparrow^0]\!]X_m & = & [\rho]X_m \\
\mathsf{whnf}\ [\rho][\![\eta]\!]X_m & = & \mathsf{whnf}\ [\rho][\![\Uparrow^0]\!](\mathsf{Lookup}\ \eta\ X_m) \\
\mathsf{whnf}\ [\rho][\![\eta]\!](\Pi A.E) & = & [\rho][\![\eta]\!](\Pi A.E) \\
\mathsf{whnf}\ [\rho][\![\eta]\!](\lambda M) & = & [\rho][\![\eta]\!](\lambda M) \\
\mathsf{whnf}\ [\rho][\![\eta]\!](M\,N) & = & (\mathsf{whnf}\ [\rho][\![\eta]\!]M)\ @\ [\rho][\![\eta]\!]N \\
\mathsf{whnf}\ [\rho][\![\eta]\!][\sigma]M & = & \mathsf{whnf}\ [\mathsf{env}\ \rho\ \eta\ \sigma][\![\eta]\!]M \\
\mathsf{whnf}\ [\rho][\![\eta]\!][\![\theta]\!]M & = & \mathsf{whnf}\ [\rho][\![\mathsf{Env}\ \eta\ \theta]\!]M
\end{array}
$$

Evaluating application $W\ @\ L$ computes the weak head normalform of $W\ L$.

$$
\begin{array}{lcl}
[\rho][\![\eta]\!](\lambda M)\ @\ L & = & \mathsf{whnf}\ [\rho, L][\![\eta]\!]M \\
H\ @\ L & = & H\ L
\end{array}
$$

Figure 4: Weak head evaluation

5. *Let* $\Delta';\Psi \vdash E : F$ *and* $\Delta;\Gamma \vdash \rho : [\![\eta]\!]\Psi$.
   If $\mathsf{whnf}\ [\rho][\![\eta]\!]E$ *is defined then* $\Delta;\Gamma \vdash \mathsf{whnf}\ [\rho][\![\eta]\!]E \equiv [\rho][\![\eta]\!]E : [\rho][\![\eta]\!]F$.
6. *Let* $\Delta;\Gamma \vdash W : \Pi A.F$ *and* $\Delta;\Gamma \vdash L : A$. *If* $W\ @\ L$ *is defined then* $\Delta;\Gamma \vdash W\ @\ L \equiv W\ L : [\uparrow^0, L]F$.

Proof. Each by induction on the trace of the function and inversion on the typing derivations, the first four statements in isolation and the remaining two simultaneously. □

3.2 Algorithmic equality

Building on the weak head normalization algorithm introduced in the previous section, we now give an algorithm for deciding equality of expressions. This is a key piece in the bi-directional type checking algorithm which we present in Section 4. Two closures, where $L = [\rho][\![\eta]\!]E$ and $L' = [\rho'][\![\eta']\!]E'$, are algorithmically equal if their weak head normal forms are related, i.e., $\mathsf{whnf}\ [\rho][\![\eta]\!]E \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ [\rho'][\![\eta']\!]E'$.

As we check that two expressions are equal, we lazily normalize them using our weak head normalization algorithm from the previous section and our algorithmic equality algorithm alternates between applying a whnf step and actually comparing two expressions or substitutions.

The actual equality algorithm is defined using three mutual recursive judgments. 1) checking that two expressions in whnf are equal 2) checking that two neutral weak head normal forms are equal and 3) checking that two environments, i.e., ordinary substitutions in whnf, are equal.

$W \overset{\mathsf{w}}{\sim} W'$	weak head normal forms W, W' are algorithmically equal
$H \overset{\mathsf{n}}{\sim} H'$	neutral weak head normal forms H, H' are algorithmically equal
$[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho'$	environments ρ, ρ' are algorithmically equal under shifts by k, k' resp.

Many of the algorithmic equality rules are straightforward and intuitive, although a bit veiled by the abundance of explicit shifting that comes with de Bruijn style. When checking whether two meta-variables are equal, we need to make sure that respective environments are equal. When we check whether two lambda-abstractions are equal, we must lift their environments under the lambda-binding. This amounts to shifting them by one and extending them with a binding for the first variable. To handle eta-equality, we eta-expand the neutral weak head normal form H on the fly when comparing it to a lambda-closure.

Comparing two environments for equality simply recursively analyzes the substitutions. In addition, we handle just-in-time eta-expansion on the level of substitutions (see the last two rules).

Algorithmic equality of neutral weak head normal forms.

$$\frac{}{a \overset{\mathsf{n}}{\sim} a} \qquad \frac{}{x_m \overset{\mathsf{n}}{\sim} x_m} \qquad \frac{[\uparrow^0]\rho \overset{\mathsf{r}}{\sim} [\uparrow^0]\rho'}{[\rho]X_m \overset{\mathsf{n}}{\sim} [\rho']X_m} \qquad \frac{H \overset{\mathsf{n}}{\sim} H' \qquad \mathsf{whnf}\ L \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ L'}{H\ L \overset{\mathsf{n}}{\sim} H'\ L'}$$

Algorithmic equality of weak head normal forms.

$$\frac{}{s \overset{\mathsf{w}}{\sim} s} \qquad \frac{\mathsf{whnf}\ [\rho][\![\eta]\!]A \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ [\rho'][\![\eta']\!]A' \qquad \mathsf{whnf}\ [[\uparrow^1]\rho, x_1][\![\eta]\!]B \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ [[\uparrow^1]\rho', x_1][\![\eta']\!]B'}{[\rho][\![\eta]\!](\Pi A.B) \overset{\mathsf{w}}{\sim} [\rho'][\![\eta']\!](\Pi A'.B')}$$

$$\frac{H \overset{\mathsf{n}}{\sim} H'}{H \overset{\mathsf{w}}{\sim} H} \qquad \frac{\mathsf{whnf}\ [[\uparrow^1]\rho, x_1][\![\eta]\!]M \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ [[\uparrow^1]\rho', x_1][\![\eta']\!]M'}{[\rho][\![\eta]\!](\lambda M) \overset{\mathsf{w}}{\sim} [\rho'][\![\eta']\!](\lambda M')}$$

$$\frac{\mathsf{whnf}\ [[\uparrow^1]\rho, x_1][\![\eta]\!]M \overset{\mathsf{w}}{\sim} (\mathsf{shift}^1\, H)\, x_1}{[\rho][\![\eta]\!](\lambda M) \overset{\mathsf{w}}{\sim} H} \qquad \frac{(\mathsf{shift}^1\, H)\, x_1 \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ [[\uparrow^1]\rho, x_1][\![\eta]\!]M}{H \overset{\mathsf{w}}{\sim} [\rho][\![\eta]\!](\lambda M)}$$

Algorithmic equality of environments.

$$\frac{k+n = k'+n'}{[\uparrow^k]\uparrow^n \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\uparrow^{n'}} \qquad \frac{[\uparrow^{k+n}]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho'}{[\uparrow^k][\uparrow^n]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho'} \qquad \frac{[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'+n'}]\rho'}{[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}][\uparrow^{n'}]\rho'}$$

$$\frac{[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho' \qquad \mathsf{whnf}\ (\mathsf{shift}^k\, L) \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ (\mathsf{shift}^{k'}\, L')}{[\uparrow^k](\rho, L) \overset{\mathsf{r}}{\sim} [\uparrow^{k'}](\rho', L')}$$

$$\frac{[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\uparrow^{n'+1} \qquad \mathsf{whnf}\ (\mathsf{shift}^k\, L) \overset{\mathsf{w}}{\sim} x_{k'+n'+1}}{[\uparrow^k](\rho, L) \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\uparrow^{n'}} \qquad \frac{[\uparrow^k]\uparrow^{n+1} \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho' \qquad x_{k+n+1} \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ (\mathsf{shift}^{k'}\, L')}{[\uparrow^k]\uparrow^n \overset{\mathsf{r}}{\sim} [\uparrow^{k'}](\rho', L')}$$

Theorem 6 (Soundness of algorithmic equality)

1. *If* $H \overset{\mathsf{n}}{\sim} H'$ *and* $\Delta;\Gamma \vdash H : F$ *and* $\Delta;\Gamma \vdash H' : F'$ *then* $\Delta;\Gamma \vdash F \equiv F'$ *and* $\Delta;\Gamma \vdash H \equiv H' : F$.
2. *If* $W \overset{\mathsf{w}}{\sim} W'$ *and* $\Delta;\Gamma \vdash W : F$ *and* $\Delta;\Gamma \vdash W' : F$ *then* $\Delta;\Gamma \vdash W \equiv W' : F$.
3. *If* $[\uparrow^k]\rho \overset{\mathsf{r}}{\sim} [\uparrow^{k'}]\rho'$ *and* $\Delta;\Gamma \vdash [\uparrow^k]\rho : \Psi$ *and* $\Delta;\Gamma \vdash [\uparrow^{k'}]\rho' : \Psi$ *then* $\Delta;\Gamma \vdash [\uparrow^k]\rho \equiv [\uparrow^{k'}]\rho' : \Psi$.

Proof. Simultaneously by induction on the derivation of algorithmic equality and inversion on the typing. □

4 Bidirectional Type Checking

In this section, we show how to use our explicit substitution calculus to type-check expressions. As mentioned in the introduction, accumulating substitution walks in type-checking is one of the key applications of this work. We only describe the algorithm and leave its theoretical properties for future work.

We design the algorithm in a bidirectional way [Coq96, AC07] which allows us to omit type annotations at lambda-abstractions. We use the following three judgments:

$\Delta;\Gamma \vdash V \leftleftarrows s$	Type normal form V checks against sort s
$\Delta;\Gamma \vdash V \leftleftarrows L$	Normal form V checks against "type" closure L
$\Delta;\Gamma \vdash U \rightrightarrows L$	The type of neutral normal form U is inferred as closure L
$\Delta;\Gamma \vdash \nu \leftleftarrows \Psi$	Normal substitution ν checks against domain Ψ

In these judgements, Γ is a list of type closures L. On Δ we pose no restrictions; an entry $\Psi \triangleright A$ of Δ is as before a list of type expressions Ψ and a type expression A.
Inferring the type of neutral normal forms U.

$$\frac{}{\Delta;\Gamma \vdash a \rightrightarrows [\uparrow^0][\![\Uparrow^0]\!]\Sigma(a)} \qquad \frac{\Delta;\Gamma \vdash U \rightrightarrows L \qquad \mathsf{whnf}\ L = [\rho][\![\eta]\!](\Pi A.B) \qquad \Delta;\Gamma \vdash V \leftleftarrows [\rho][\![\eta]\!]A}{\Delta;\Gamma \vdash U\ V \rightrightarrows [\rho, V][\![\eta]\!]B}$$

$$\frac{|\Gamma'| = n}{\Delta;\Gamma, L, \Gamma' \vdash x_{n+1} \rightrightarrows \mathsf{shift}^{n+1}\, L} \qquad \frac{\Delta = \Delta_1, \Psi \triangleright A, \Delta_2 \qquad |\Delta_2| = n \qquad \Delta;\Gamma \vdash \nu \leftleftarrows [\uparrow^0][\![\Uparrow^{n+1}]\!]\Psi}{\Delta;\Gamma \vdash [\nu]X_{n+1} \rightrightarrows [\nu][\![\Uparrow^{n+1}]\!]A}$$

Checking the type of normal forms V.

$$\frac{\mathsf{whnf}\ L = [\rho][\![\eta]\!](\Pi A.B) \quad \Delta;\Gamma,[\rho][\![\eta]\!]A \vdash V \leftleftarrows [\uparrow^1\rho, x_1][\![\eta]\!]B}{\Delta;\Gamma \vdash \lambda V \leftleftarrows L} \qquad \frac{\Delta;\Gamma \vdash U \rightrightarrows L \quad \mathsf{whnf}\ L \overset{\mathsf{w}}{\sim} \mathsf{whnf}\ L'}{\Delta;\Gamma \vdash U \leftleftarrows L'}$$

Checking well-formedness of types and kinds V.

$$\frac{}{\Delta;\Gamma \vdash \mathsf{type} \leftleftarrows \mathsf{kind}} \qquad \frac{\Delta;\Gamma \vdash V \leftleftarrows \mathsf{type} \quad \Delta;\Gamma,[\uparrow^0][\![\Uparrow^0]\!]V \vdash V' \leftleftarrows s}{\Delta;\Gamma \vdash \Pi V.V' \leftleftarrows s} \qquad \frac{\Delta;\Gamma \vdash U \rightrightarrows L \quad \mathsf{whnf}\ L = \mathsf{type}}{\Delta;\Gamma \vdash U \leftleftarrows \mathsf{type}}$$

Checking normal substitutions ν. In this judgement $\Delta;\Gamma \vdash \nu \leftleftarrows \Psi$, the context Ψ is also in closure form.

$$\frac{|\Gamma| = n}{\Delta;\Gamma \vdash \uparrow^n \leftleftarrows \cdot} \qquad \frac{\Delta;\Gamma \vdash \nu \leftleftarrows \Psi \qquad \Delta;\Gamma \vdash V \leftleftarrows L}{\Delta;\Gamma \vdash (\nu, V) \leftleftarrows \Psi, L}$$

5 Conclusion

We have presented an explicit substitution calculus together with algorithms for weak head normalization, definitional equality, and bi-directional type checking where both ordinary variables and meta-variables are modelled using de Bruijn indices and both kinds of substitutions are handled lazily and simultaneously.

We also have proven subject reduction and soundness of the definitional equality algorithm. Finally, we describe a bi-directional type-checking algorithm which treats ordinary substitutions and meta-substitutions at the same time. In the future, we plan to prove completeness of algorithmic equality and type checking and to adapt the presented explicit substitutions in the implementation of the programming and reasoning environment Beluga.

References

[AC07] Andreas Abel and Thierry Coquand. Untyped algorithmic equality for Martin-Löf's logical framework with surjective pairs. *Fundamenta Informaticae*, 77(4):345–395, 2007. TLCA'05 special issue.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Ada05] Robin Adams. *A Modular Hierarchy of Logical Frameworks*. PhD thesis, University of Manchester, 2005.

[BDN09] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - a functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs'09)*, volume 5674 of *Lecture Notes in Computer Science*, pages 73–78. Springer-Verlag, 2009.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[Coq96] Thierry Coquand. An algorithm for type-checking dependent types. In *Mathematics of Program Construction. Selected Papers from the Third International Conference on the Mathematics of Program Construction (July 17–21, 1995, Kloster Irsee, Germany)*, volume 26 of *Science of Computer Programming*, pages 167–177. Elsevier, May 1996.

[DHK00] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher order unification via explicit substitutions. *Information and Computation*, 157(1-2):183–235, 2000.

[HP05] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic*, 6(1):61–101, 2005.

[LNQ05] Chuck Liang, Gopalan Nadathur, and Xiaochu Qi. Choices in representation and reduction strategies for lambda terms in intensional contexts. *Journal of Automated Reasoning*, 33(2):89–132, 2005.

[Nor07] Ulf 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, September 2007.

[NPP08] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

[NW98] Gopalan Nadathur and Debra Sue Wilson. A notation for lambda terms: A generalization of environments. *Theoretical Computer Science*, 198(1-2):49–98, 1998.

[PD08] Brigitte Pientka and Joshua Dunfield. Programming with proofs and explicit contexts. In *ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'08)*, pages 163–173. ACM Press, July 2008.

[PD10] Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In Jürgen Giesl and Reiner Hähnle, editors, *5th International Joint Conference on Automated Reasoning (IJCAR'10)*, Lecture Notes in Computer Science. Springer-Verlag, 2010.

[Pie03] Brigitte Pientka. *Tabled higher-order logic programming*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 2003. CMU-CS-03-185.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 202–206. Springer, 1999.

[PS08] Adam B. Poswolsky and Carsten Schürmann. Practical programming with higher-order encodings and dependent types. In *Proceedings of the 17th European Symposium on Programming (ESOP '08)*, volume 4960, page 93. Springer, March 2008.

[PS09] Adam Poswolsky and Carsten Schürmann. System description: Delphin - a functional programming language for deductive systems. *Electronic Notes in Theoretical Computer Science*, 228:113–120, 2009.