Contextual Modal Type Theory

ALEKSANDAR NANEVSKI
Harvard University
and
FRANK PFENNING
Carnegie Mellon University
and
BRIGITTE PIENTKA
McGill University

The intuitionistic modal logic of necessity is based on the judgmental notion of categorical truth. In this paper we investigate the consequences of relativizing these concepts to explicitly specified contexts. We obtain contextual modal logic and its type-theoretic analogue. Contextual modal type theory provides an elegant, uniform foundation for understanding meta-variables and explicit substitutions. We sketch some applications in functional programming and logical frameworks.

Categories and Subject Descriptors: F.4.1 [**Theory of Computation**]: Mathematical Logic and Formal Languages—*Modal Logic*; D.3.3 [**Software**]: Language Constructs and Features—*Frameworks*

General Terms: Design, Theory

Additional Key Words and Phrases: Type theory, logical frameworks, intuitionistic modal logic

1. INTRODUCTION

The dictionary[1] defines *context* as "*the parts of a discourse that surround a word or passage and can throw light on its meaning*" and also as "*the interrelated conditions in which something exists or occurs*". One can see from these definitions that the notion of context, with somewhat differentiated meanings, is fundamental in linguistics, artificial intelligence, and logic. Narrowing our scope to logic, the notion of context enters almost immediately because the truth of a proposition is not absolute, but depends on the context we consider it in.

Taking the next step, namely allowing the propositions in a logic itself to speak about contexts, is the subject of modal logic and fraught with many difficulties and paradoxes. But it is also very important. The study of contexts in logic has ram-

[1]Merriam-Webster Online

This material is based upon work supported by the National Science Foundation under Grant No. 0325808.

ifications for topics such as modularity, reflection, spatial and temporal reasoning, and meta-reasoning. Because of the close relationship between constructive logic and programming languages, contexts are also directly relevant to computational phenomena, particularly data abstraction, run-time code generation, distributed computation, partial evaluation, and meta-programming.

Narrowing our scope a bit further, in this paper we think of a context as consisting of the hypotheses we make while trying to establish a conclusion. Contexts are therefore intrinsically tied to hypothetical judgments. We investigate how to internalize reasoning about contexts within the logic while retaining an intuitionistic and predicative viewpoint regarding the meaning of the logical connectives. In this manner we can give our core calculus several interesting interpretations. One, in the realm of functional programming, relates contexts to intensional expressions and run-time code generation in the presence of free variables. Another, in the realm of logic programming, explains explicit substitutions and meta-variables. In each case, the strength of the underlying logical foundations provides new insights for language design which has so far been largely motivated operationally. We return to these applications in Sections 6 and 7.

There seem to be two main approaches to intuitionistic modal logic. One, which we call *nominal*, assigns names to contexts and includes explicit judgments to relate them. This can be seen as the intuitionistic analogue of Kripke's classical multiple-world semantics [Kripke 1959] and has been investigated in some depth by Simpson [1994] and applied to partial evaluation [Davies 1996] and distributed computation [Murphy VII et al. 2004]. We will not pursue the nominal approach here. The other, which we call *structural*, identifies the context with the propositions it contains. Pfenning and Davies [2001] present the first exploration in this direction, singling out the empty context which represents reasoning without truth assumptions. This provides a constructive meaning explanation for necessity, possibility, and lax truth. In this paper we carry the structural approach significantly further, allowing arbitrary contexts to be captured within a proposition. This sheds new light on some important ideas. For example, *explicit substitutions* [Cartmell 1986; Abadi et al. 1990] are inevitable as evidence that we can reach one context from another. Another example is *meta-variables* which arise as we introduce hypotheses about the truth of propositions in given contexts.

We start our investigation in the simplest case, namely a propositional logic with a contextual modality. We develop both natural deduction (Section 2) and the sequent calculus (Section 3). In view of applications in computer science, we then generalize to a type theory in Section 4, which differs from the logic in that it carries explicit proof terms. We then generalize further to add dependent types (Section 5) and show applications to staged functional programming (Section 6) and logical frameworks (Section 7). We close with a discussion of related and future work.

2. INTUITIONISTIC CONTEXTUAL MODAL LOGIC

The philosophical foundation of our development is Martin-Löf's approach of separating judgments from propositions [Martin-Löf 1996]. The most basic judgment is the *truth* of a *proposition*, written as A true. We explain the meaning of a propo-

sition by presenting the means of inferring its truth via introduction rules, and for exploiting the knowledge of its truth via elimination rules. The construction of propositions in this manner is inherently open-ended, so we do not circumscribe all propositions. In this section we assume there are some atomic propositions P and we define implication $A \to B$ and a (contextual) modal operator $[\Psi]A$ where Ψ is a context.

2.1 Hypothetical Judgments

In order to explain the meaning of implication and later of the modal operators in this manner, we need the notion of a *hypothetical judgment*. We write

$$x_1{:}A_1\ \mathsf{true}, \ldots, x_n{:}A_n\ \mathsf{true} \vdash A\ \mathsf{true}$$

to express that A is true whenever all hypotheses $A_1, \ldots, A_n$ are true. We always label hypotheses with distinct variables x_i in order to avoid any ambiguity in proofs. The properties of hypothetical judgments are captured by the hypothesis rule and the substitution principle. We abbreviate a collection of assumptions by Γ. Even though at the moment it might be convenient to view contexts as unordered, later contexts are reified in modal operators and may even be internally dependent, at which point order is important. We therefore view them as lists, where comma ',' is overloaded in the usual way to add a new element at the end of a context or to append two contexts.

Hypothesis Rule.

$$\frac{}{\Gamma, x{:}A\ \mathsf{true}, \Gamma' \vdash A\ \mathsf{true}}\ \mathsf{hyp}_x$$

Note that without distinct labels for assumptions, this rule would be ambiguous.

Substitution Principle.

If $\Gamma \vdash A\ \mathsf{true}$ *and* $\Gamma, x{:}A\ \mathsf{true}, \Gamma' \vdash C\ \mathsf{true}$ *then* $\Gamma, \Gamma' \vdash C\ \mathsf{true}$.

While the hypothesis rule is primitive, the substitution principle should always be *admissible*: whenever we have a proof of $A\ \mathsf{true}$ then we can substitute that proof for all uses of an assumption $A\ \mathsf{true}$ in another proof. We will not further elaborate here on the standard notion of hypothetical judgments.

Now we can explain the meaning of implication.

Implication Introduction. $A \to B$ is true if B is true under the hypothesis that A is true.

$$\frac{\Gamma, x{:}A\ \mathsf{true} \vdash B\ \mathsf{true}}{\Gamma \vdash A \to B\ \mathsf{true}}\ \to\!\mathsf{I}^x$$

By our convention regarding well-formed contexts, x must not already be declared in Γ.

Implication Elimination. Conversely, if we know that $A \to B$ is true, then B is true under hypothesis that A is true. By the substitution principle we should

therefore be able to obtain a proof of B from a proof of A.

$$\frac{\Gamma \vdash A \to B\ \mathsf{true} \quad \Gamma \vdash A\ \mathsf{true}}{\Gamma \vdash B\ \mathsf{true}}\ {\to}\mathsf{E}$$

In order to check that the elimination rules are not too strong, we should verify that any introduction of a connective immediately followed by its elimination can be reduced. This means the elimination rule cannot extract more knowledge from a proposition than contributed by its proof. In other words, it is *sound*. Local reductions also give rise to computation in the setting of functional programming.

Local Reduction for Implication.

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, x{:}A\ \mathsf{true} \vdash B\ \mathsf{true}\end{array}}{\Gamma \vdash A \to B\ \mathsf{true}}\ {\to}\mathsf{I}^x \qquad \begin{array}{c}\mathcal{E}\\ \Gamma \vdash A\ \mathsf{true}\end{array}}{\Gamma \vdash B\ \mathsf{true}}\ {\to}\mathsf{E} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{D}'\\ \Gamma \vdash B\ \mathsf{true}\end{array}$$

Here, $\mathcal{D}'$ can be obtained from $\mathcal{D}$ and $\mathcal{E}$ by application of the substitution principle. Computationally, this corresponds to a β-reduction.

In order to check that the elimination rules are not too weak, we should verify that we can recover the knowledge that contributed to the proof of a proposition. In other words, starting from any proof of a proposition, we can apply the elimination rules in such a way that we can reintroduce the proposition from the results. Local expansions correspond to extensionality in functional programming.

Local Expansion for Implication.

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \to B\ \mathsf{true}\end{array} \quad \Longrightarrow_E \quad \frac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \Gamma, x{:}A\ \mathsf{true} \vdash A \to B\ \mathsf{true}\end{array} \quad \dfrac{}{\Gamma, x{:}A\ \mathsf{true} \vdash A\ \mathsf{true}}\ \mathsf{hyp}_x}{\Gamma, x{:}A\ \mathsf{true} \vdash B\ \mathsf{true}}\ {\to}\mathsf{E}}{\Gamma \vdash A \to B\ \mathsf{true}}\ {\to}\mathsf{I}^x$$

Here, $\mathcal{D}'$ can be obtained from $\mathcal{D}$ by *weakening*, that is, adjoining the hypothesis $x{:}A\,\mathsf{true}$ to every judgment in $\mathcal{D}$. In functional programming, local expansion corresponds to an η-expansion.

2.2 Categorical Judgments

In this section we review the critical notion of a *categorical judgment* [Pfenning and Davies 2001]. We say that A is *categorically true* or *valid* if its truth does not depend on any hypotheses about the truth of other propositions. However, it may depend on the validity of other proposition, because their validity may not depend on truth assumptions.

We write Δ for a labeled list $u_1{::}A_1\ \mathsf{valid}, \ldots, u_k{::}A_k\ \mathsf{valid}$ of assumptions about the validity of propositions, using "$\cdot$" for an empty list. With this notation we separate assumptions about validity and truth and write

$$\Delta; \Gamma \vdash J$$

where J is either $A\ \mathsf{true}$ or $A\ \mathsf{valid}$.

Definition of Validity.

$$\frac{\Delta; \cdot \vdash A\ \mathsf{true}}{\Delta; \Gamma \vdash A\ \mathsf{valid}}$$

Here the premise represents a truth-categorical judgment. Conversely, if A is valid then we can conclude it is true.

$$\frac{}{(\Delta, u{::}A\,\mathsf{valid}, \Delta'); \Gamma \vdash A\ \mathsf{true}}\ \mathsf{vldhyp}_u$$

Another way to think about this definition is in terms of a multiple-world semantics. We say that A is valid if A is true in every possible world. Since we can never circumscribe all possible worlds, we instead require that A must be true in a world about which we assume nothing ($\Gamma = \cdot$), except for the propositions we assumed to be always true (Δ).

Note that by the definition of validity we can immediately reduce the goal of proving A valid to A true. We can always apply this silently and thereby remove the judgment A valid from consideration as the conclusion of a hypothetical judgment. No such trick applies to assumptions of the form A valid, so we require a new substitution principle.

Substitution Principle for Validity.

If $\Delta; \cdot \vdash A$ true *and* $(\Delta, u{::}A\,\mathsf{valid}, \Delta'); \Gamma \vdash C$ true
then $(\Delta, \Delta'); \Gamma \vdash C$ true.

Note that the first assumption expresses $\Delta; \Gamma \vdash A$ valid, thereby justifying the substitution principle directly from the nature of hypothetical judgments in general.

Using the categorical judgment, it is now possible to introduce a modal operator for necessary truth, $\Box A$. Since we need a more general operator we forego this analysis here; it can be found in the paper by Pfenning and Davies [2001]. The result is intuitionistic S4 which captures reflexivity and transitivity of accessibility between worlds in its nominal formulation [Simpson 1994].

2.3 Contextual Validity

The judgment A valid expresses that A is true in any world. In this section we relativize this judgment: let Ψ be a list of assumptions $y_1{:}B_1\,\mathsf{true}, \ldots, y_m{:}B_m\,\mathsf{true}$. We say that A *is valid relative* Ψ (written as $A\ \mathsf{valid}[\Psi]$) if A is true in every world in which $B_1, \ldots, B_m$ are true. We similarly generalize assumptions about validity so that Δ now has the form $u_1{::}A_1\,\mathsf{valid}[\Psi_1], \ldots, u_k{::}A_k\,\mathsf{valid}[\Psi_k]$. Note that $A_i\,\mathsf{valid}[\Psi_i]$ is not a proposition, only notation of a judgment. A corresponding proposition is introduced in Section 2.4.

This generalization does not directly add expressive power, since a judgment of the general form $A\ \mathsf{valid}[y_1{:}B_1\mathsf{true}, \ldots, y_n{:}B_m\,\mathsf{true}]$ holds iff $(B_1 \to \cdots (B_m \to A))$ valid. However, there is an important difference in the proof theory, which we exploit in several applications to obtain context calculi for staged computation and meta-variables.

As common in programming languages, we refer to Ψ as a *context* and to the judgment $A\ \mathsf{valid}[\Psi]$ as *contextual validity*.

Definition of Contextual Validity. In order to prove $A\ \mathsf{valid}[\Psi]$ we have to prove A using only Ψ and without any other current hypotheses about truth. Assumptions about contextual validity, however, carry over as before.

$$\frac{\Delta; \Psi \vdash A\ \mathsf{true}}{\Delta; \Gamma \vdash A\ \mathsf{valid}[\Psi]}$$

As in the case of ordinary validity in Section 2.2, proving $\Delta; \Gamma \vdash A\ \mathsf{valid}[\Psi]$ can always be immediately reduced to $\Delta; \Psi \vdash A\ \mathsf{true}$. Therefore we are justified in streamlining our analysis by not considering validity on the right-hand side of a hypothetical judgment

Contextual Entailment. We can exploit the knowledge that $A\ \mathsf{valid}[\Psi]$ to conclude that $A\ \mathsf{true}$, but only if we can show that all propositions in Ψ follow from the current assumptions. We write

$$\Delta; \Gamma \vdash \Psi$$

to express that all the propositions in Ψ are true using the assumptions from Γ.

$$\frac{\Delta; \Gamma \vdash B_1\ \mathsf{true} \quad \ldots \quad \Delta; \Gamma \vdash B_m\ \mathsf{true}}{\Delta; \Gamma \vdash y_1{:}B_1\ \mathsf{true}, \ldots, y_m{:}B_m\ \mathsf{true}}\ \mathsf{ctx}$$

Contextual Hypothesis Rule. Now we can state the contextual hypothesis rule.

$$\frac{(\Delta, u{::}A\ \mathsf{valid}[\Psi], \Delta'); \Gamma \vdash \Psi}{(\Delta, u{::}A\ \mathsf{valid}[\Psi], \Delta'); \Gamma \vdash A\ \mathsf{true}}\ \mathsf{ctxhyp}_u$$

In order to use the assumption that A is valid relative Ψ, we must show that our current hypotheses in Γ are strong enough to establish Ψ.

With the contextual hypothesis rule in place, it is easy to verify that the judgment $A\ \mathsf{valid}$ from the previous section is recovered as $A\ \mathsf{valid}[\cdot]$. Of course, this is a special case of contextual validity where the relevant context is taken to be empty. For example, the instance of the contextual hypothesis rule

$$\frac{\dfrac{}{(\Delta, u{::}A\ \mathsf{valid}[\cdot], \Delta'); \Gamma \vdash \cdot}\ \mathsf{ctx}}{(\Delta, u{::}A\ \mathsf{valid}[\cdot], \Delta'); \Gamma \vdash A\ \mathsf{true}}\ \mathsf{ctxhyp}_u$$

has no premises because the empty context "$\cdot$" has no members and so becomes the modal hypothesis rule vldhyp_u.

Contextual Substitution Principle. The substitution principle generalizes in a straightforward way.

> *If* $\Delta; \Psi \vdash A\ \mathsf{true}$ *and* $(\Delta, u{::}A\ \mathsf{valid}[\Psi], \Delta'); \Gamma \vdash C\ \mathsf{true}$
> *then* $(\Delta, \Delta'); \Gamma \vdash C\ \mathsf{true}$.

The first condition expresses $\Delta; \Gamma \vdash A\ \mathsf{valid}[\Psi]$. The principle is therefore justified by the general substitution property of hypothetical judgments. The associated substitution operation substitutes the proof of A under Ψ for uses of the assumption that A is valid under Ψ. A more precise description is possible when we add proof terms in Section 4.

Contextual Identity Principle. One particular instance of contextual entailment combined with the ordinary hypothesis rule is the contextual identity principle.

$$\Delta; \Psi \vdash \Psi$$

This holds because each conclusion B_j true is proved in one step from the corresponding assumption $y_j{:}B_j$ true by the hypothesis rule applied to y_j. Like the substitution principle, this is not a primitive rule, but every instance of it is derivable in our system.

2.4 Contextual Modal Necessity

A modal logic arises from the judgmental definition of contextual validity if we internalize the judgment as a modal propositional operator. We write $[\Psi]A$ for the *proposition* asserting that A is valid in context Ψ. The difference is of course that a proposition can appear as part of other propositions, while judgments cannot be combined with propositional connectives.

Necessity Introduction. The introduction rule is straightforward, simply unfolding the definition of validity.

$$\frac{\Delta; \Psi \vdash A \text{ true}}{\Delta; \Gamma \vdash [\Psi]A \text{ true}}\ \Box\mathsf{I}$$

Necessity Elimination. The elimination rule is based on the substitution principle: if we can prove $[\Psi]A$ then we are justified in assuming A valid$[\Psi]$.

$$\frac{\Delta; \Gamma \vdash [\Psi]A \text{ true} \quad (\Delta, u{::}A \text{ valid}[\Psi]); \Gamma \vdash C \text{ true}}{\Delta; \Gamma \vdash C \text{ true}}\ \Box\mathsf{E}^u$$

As usual, we assume that u is not already declared in Δ so that all hypotheses have distinct labels.

Local Reduction. We have to show that an introduction of necessity followed immediately by its elimination can be reduced, avoiding the detour.

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Psi \vdash A \text{ true}\end{array}}{\Delta; \Gamma \vdash [\Psi]A \text{ true}}\ \Box\mathsf{I} \qquad \begin{array}{c}\mathcal{E}\\ (\Delta, u{::}A \text{ valid}[\Psi]); \Gamma \vdash C \text{ true}\end{array}}{\Delta; \Gamma \vdash C \text{ true}}\ \Box\mathsf{E}^u \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{E}'\\ \Delta; \Gamma \vdash C \text{ true}\end{array}$$

Here, $\mathcal{E}'$ is obtained from $\mathcal{E}$ by application of the contextual substitution principle, replacing uses of the assumption u with the proof $\mathcal{D}$.

Local Expansion. We have to show that we can apply the elimination rule in such a way that we can reconstitute a proof of $[\Psi]A$ from the result.

$$\frac{}{\Delta;(\Gamma,x{:}A\,\mathsf{true},\Gamma')\vdash A\,\mathsf{true}}\;\mathsf{hyp}_x \qquad \frac{(\Delta,u{::}A\,\mathsf{valid}[\Psi],\Delta');\Gamma\vdash\Psi}{(\Delta,u{::}A\,\mathsf{valid}[\Psi],\Delta');\Gamma\vdash A\,\mathsf{true}}\;\mathsf{ctxhyp}_u$$

$$\frac{\Delta;\Gamma,x{:}A\,\mathsf{true}\vdash B\,\mathsf{true}}{\Delta;\Gamma\vdash A\to B\,\mathsf{true}}\;{\to}\mathsf{I}^x \qquad \frac{\Delta;\Gamma\vdash A\to B\,\mathsf{true}\quad\Delta;\Gamma\vdash A\,\mathsf{true}}{\Delta;\Gamma\vdash B\,\mathsf{true}}\;{\to}\mathsf{E}$$

$$\frac{\Delta;\Psi\vdash A\,\mathsf{true}}{\Delta;\Gamma\vdash[\Psi]A\,\mathsf{true}}\;\Box\mathsf{I} \qquad \frac{\Delta;\Gamma\vdash[\Psi]A\,\mathsf{true}\quad\Delta,u{::}A\,\mathsf{valid}[\Psi];\Gamma\vdash C\,\mathsf{true}}{\Delta;\Gamma\vdash C\,\mathsf{true}}\;\Box\mathsf{E}^u$$

$$\frac{\Delta;\Gamma\vdash B_1\,\mathsf{true}\quad\ldots\quad\Delta;\Gamma\vdash B_m\,\mathsf{true}}{\Delta;\Gamma\vdash y_1{:}B_1\,\mathsf{true},\ldots,y_m{:}B_m\,\mathsf{true}}\;\mathsf{ctx}$$

Fig. 1. Natural Deduction for ICML

$$\overset{\mathcal{D}}{\Delta;\Gamma\vdash[\Psi]A\,\mathsf{true}}\;\Longrightarrow_E\;\frac{\overset{\mathcal{D}}{\Delta;\Gamma\vdash[\Psi]A\,\mathsf{true}}\quad\dfrac{\dfrac{\overset{\mathcal{E}}{(\Delta,u{::}A\,\mathsf{valid}[\Psi]);\Psi\vdash\Psi}}{(\Delta,u{::}A\,\mathsf{valid}[\Psi]);\Psi\vdash A\,\mathsf{true}}\;\mathsf{ctxhyp}_u}{(\Delta,u{::}A\,\mathsf{valid}[\Psi]);\Gamma\vdash[\Psi]A\,\mathsf{true}}\;\Box\mathsf{I}}{\Delta;\Gamma\vdash[\Psi]A\,\mathsf{true}}\;\Box\mathsf{E}^u$$

Here the derivation $\mathcal{E}$ exists by the contextual identity principle. This is analogous to an appeal to the contextual substitution principle during local reduction.

This particular local expansion result has an unusual shape in that the last rule on the right-hand side is an elimination, not an introduction as in the local expansion for implication. Reading it as: "*Apply the elimination rule to the proof* $\mathcal{D}$ *of* $[\Psi]A$ true*, call the result* u*, and then reconstitute a proof of* $[\Psi]A$ true *from* u." should illustrate that this is just a notational oddity of the $\Box\mathsf{E}$ and similar elimination rules which introduce new assumptions.

We call the resulting logic intuitionistic contextual modal logic (ICML). The rules of natural deduction are summarized in Figure 1.

2.5 Examples

Before presenting more of the theory of intuitionistic contextual modal logic, we give some example derivations. We follow the usual convention that the modal operator (here $[\Psi]$) binds more tightly than the other logical connectives and that "$\to$" associates to the right. Where the names of the variables in contexts Ψ do not matter we abbreviate $x{:}A\,\mathsf{true}$ simply by A.

(1) $\vdash [C]A \to [C,D]A\,\mathsf{true}$
(2) $\vdash [C,C]A \to [C]A\,\mathsf{true}$
(3) $\vdash [A]A\,\mathsf{true}$
(4) $\vdash [A]B \to [A][B]C \to [A]C\,\mathsf{true}$
(5) $\vdash [\cdot]A \to A\,\mathsf{true}$
(6) $\vdash [C]A \to [D][C]A\,\mathsf{true}$

(7) $\vdash [C](A \to B) \to ([D]A \to [C, D]B)$ true
(8) $\vdash [A](A \to B) \to ([B]C \to [A]C)$ true
(9) $\not\vdash A \to [\cdot]A$ true
(10) $\not\vdash A \to [B]A$ true
(11) $\not\vdash (A \to B) \to [B]C \to [A]C$ true

The derivation of (1) closely mirrors the local expansion for the type $[\Psi]A$, as presented in the previous section.

$$\dfrac{\dfrac{}{x{:}[C]A\,\mathsf{true} \vdash [C]A\,\mathsf{true}}\,\mathsf{hyp}_x \qquad \dfrac{\dfrac{\dfrac{}{u{::}A\,\mathsf{valid}[C]; x{:}C\,\mathsf{true}, y{:}D\,\mathsf{true} \vdash C\,\mathsf{true}}\,\mathsf{hyp}_x}{u{::}A\,\mathsf{valid}[C]; x{:}C\,\mathsf{true}, y{:}D\,\mathsf{true} \vdash C}\,\mathsf{ctx}}{u{::}A\,\mathsf{valid}[C]; x{:}C\,\mathsf{true}, y{:}D\,\mathsf{true} \vdash A\,\mathsf{true}}\,\mathsf{ctxhyp}_u}{u{::}A\,\mathsf{valid}[C]; x{:}[C]A \vdash [C, D]A\,\mathsf{true}}\,\Box\mathsf{I}}{x{:}[C]A\,\mathsf{true} \vdash [C, D]A\,\mathsf{true}}\,\Box\mathsf{E}^u}{\vdash [C]A \to [C, D]A\,\mathsf{true}}\,{\to}\mathsf{I}^x$$

We now turn to the derivation of (5). The proof of (5) shows that a proposition derived relative to an empty context is actually unconditionally true, and can thus be obtained in an arbitrary context.

$$\dfrac{\dfrac{\dfrac{}{x{:}[\cdot]A\,\mathsf{true} \vdash [\cdot]A\,\mathsf{true}}\,\mathsf{hyp}_x \qquad \dfrac{\dfrac{}{u{::}A\,\mathsf{valid}[\cdot]; \cdot \vdash \cdot}\,\mathsf{ctx}}{u{::}A\,\mathsf{valid}[\cdot]; \cdot \vdash A\,\mathsf{true}}\,\mathsf{ctxhyp}_u}{x{:}[\cdot]A\,\mathsf{true} \vdash A\,\mathsf{true}}\,\Box\mathsf{E}^u}{\vdash [\cdot]A \to A\,\mathsf{true}}\,{\to}\mathsf{I}^x$$

We omit the remaining derivations. On the other hand, the last three are not derivable in general. We do not yet have the tools to prove this, but we may attempt the proof of (9) from the bottom up.

$$\dfrac{\dfrac{\vdash A\,\mathsf{true}}{x{:}A\,\mathsf{true} \vdash [\cdot]A\,\mathsf{true}}\,\Box\mathsf{I}}{\vdash A \to [\cdot]A\,\mathsf{true}}\,{\to}\mathsf{I}^x$$

As can be seen, this approach leaves us with the generally unattainable subgoal of proving A true in an empty context. The fact that $A \to [\cdot]A$ true is indeed unprovable will be an immediate consequence of the completeness of the cut-free sequent calculus introduced in Section 3.

2.6 Elementary Properties

In this section we formally prove some properties of intuitionistic contextual modal logic on the connectives we have introduced so far. The main normal form theorem

is postponed to Section 4.5; here we concentrate on other structural properties.

Since we have a separate judgment for contextual entailment, we need to state several properties for a conclusion $A\ \mathsf{true}$ but also for the conclusion Ψ. We use J to stand for either of these two judgments. We begin with weakening and two simple properties of contextual entailment. We will often use the fundamental properties tacitly.

THEOREM 2.1 WEAKENING, IDENTITY, AND EXTENSION.

(1) ***(Weakening)*** *If* $\Delta;\Gamma \vdash J$ *then* $\Delta;(\Gamma, x{:}A\,\mathsf{true}, \Gamma') \vdash J$ *and* $(\Delta, u{::}A\,\mathsf{valid}[\Psi], \Delta');\Gamma \vdash J$.

(2) ***(Identity)*** $\Delta;\Psi \vdash \Psi$.

(3) ***(Extension)*** *If* $\Delta;\Gamma \vdash \Psi$ *then* $\Delta;(\Gamma, x{:}A\,\mathsf{true}) \vdash (\Psi, x{:}A\,\mathsf{true})$.

PROOF. By straightforward inductions. □

Next, the substitution properties, some of which we already stated as design principles above.

THEOREM 2.2 SUBSTITUTION.

(1) ***(Substitution)*** *If* $\Delta;\Gamma \vdash A\ \mathsf{true}$ *and* $\Delta;(\Gamma, x{:}A\,\mathsf{true}, \Gamma') \vdash J$ *then* $\Delta;(\Gamma,\Gamma') \vdash J$.

(2) ***(Contextual Substitution)*** *If* $\Delta;\Psi \vdash A$ *and* $(\Delta, u{::}A\ \mathsf{valid}[\Psi], \Delta');\Gamma \vdash J$ *then* $(\Delta,\Delta');\Gamma \vdash J$.

(3) ***(Simultaneous Substitution)*** *If* $\Delta;\Gamma \vdash \Psi$ *and* $\Delta;\Psi \vdash J$ *then* $\Delta;\Gamma \vdash J$.

PROOF. For substitution and contextual substitution by induction over the second given derivation. For simultaneous substitution by induction over the second given derivation where we augment the first given derivation using extension whenever necessary to apply the induction hypothesis. □

The only property we have not yet discussed is simultaneous substitution, so named because it eliminates all assumptions in the context Ψ in one operation. If J is itself a context it corresponds to composition of two substitutions, as will become clear when we make proof terms explicit. Otherwise it is simply the application of a simultaneous substitution to a term.

3. VERIFICATIONS

The system of natural deduction presented so far provides a meaning explanation of the connectives in terms of rules of proof. As a preliminary check we have ascertained that, locally, the introduction of a connective followed by its elimination can be reduced. The corresponding global property means that we can always understand the meaning of a proposition by examining only its constituents. More specifically, to accept the meaning explanations as sound we require each true proposition to have a *verification* which only refers to constituent propositions. In general, a proof may not have this property, and the existence of local reductions by themselves is insufficient to guarantee it.

We therefore develop a sequent calculus which is based on two judgments: A has a verification (written $A\ \mathsf{verif}$) and A is a hypothesis (written $A\,\mathsf{hyp}$). Ignoring, for

the moment, validity, our form of hypothetical judgment is

$$x_1{:}A_1\ \mathsf{hyp}, \ldots, x_n{:}A_n\ \mathsf{hyp} \Longrightarrow C\ \mathsf{verif}$$

where, as usual, all variables labeling hypotheses must be distinct. We refer to this judgment form as a *sequent*. The derivation of a sequent will serve as its *verification* and must therefore mention only propositions already in the sequent.

These two new judgments are connected explicitly via an init rule and two principles which will always be admissible: *identity* and *cut*. We give these three first in their simplest from, ignoring modalities.

Initial Sequents. First we state that we are willing to accept a hypothesis as a verification when the proposition is atomic and can not be decomposed further. We write P for atomic propositions.

$$\frac{}{\Gamma, x{:}P\ \mathsf{hyp}, \Gamma' \Longrightarrow P\ \mathsf{verif}}\ \mathsf{init}_x$$

Note that because judgments about P on the left- and right-hand side are different, this is not a consequence of the general definition of hypothetical judgments, but is an explicit rule relating two judgments.

Identity Principle. There is a verification of any proposition from itself as an assumption.

$$\Gamma, x{:}A\ \mathsf{hyp}, \Gamma' \Longrightarrow A\ \mathsf{verif}\ \textit{for any proposition } A.$$

Identity is not a rule in our system, since a verification of A should always analyze its structure.

Cut Principle. The principle of cut states that if we have a verification of A then we are justified in making the assumption $A\,\mathsf{hyp}$.

$$\textit{If}\ \Gamma \Longrightarrow A\ \mathsf{verif}\ \textit{and}\ \Gamma, x{:}A\ \mathsf{hyp}, \Gamma' \Longrightarrow C\ \mathsf{verif}\ \textit{then}\ \Gamma, \Gamma' \Longrightarrow C\ \mathsf{verif}.$$

Cut is not a rule in our system, since it would destroy the very nature of verifications: the proposition A need not occur in the conclusion, but would appear in both premises.

3.1 Sequent Calculus for Contextual Modal Logic

Given the system of natural deduction, it is easy to generalize the ideas above to a sequent calculus that incorporates verifications both for truth and validity. Following standard convention, we do not explicitly mention judgments and labels of hypotheses, identifying them instead by their position in sequents. Sequents then have the form

$$\Delta; \Gamma \Longrightarrow C$$

asserting that C has a verification, and

$$\Delta; \Gamma \Longrightarrow \Psi$$

asserting that each proposition in Ψ has a verification. Moreover, we take the liberty of writing Γ, A and $\Delta, A[\Psi]$ if A or $A[\Psi]$ occurs anywhere in Γ or Δ, respectively.

$$\frac{}{\Delta;\Gamma,P\Longrightarrow P}\ \mathsf{init} \qquad \frac{\Delta,A[\Psi];\Gamma\Longrightarrow\Psi \quad \Delta,A[\Psi];\Gamma,A\Longrightarrow C}{\Delta,A[\Psi];\Gamma\Longrightarrow C}\ \mathsf{reflect}$$

$$\frac{\Delta;\Gamma,A\Longrightarrow B}{\Delta;\Gamma\Longrightarrow A\to B}\ {\to}\mathsf{R} \qquad \frac{\Delta;\Gamma,A\to B\Longrightarrow A \quad \Delta;\Gamma,A\to B,B\Longrightarrow C}{\Delta;\Gamma,A\to B\Longrightarrow C}\ {\to}\mathsf{L}$$

$$\frac{\Delta;\Psi\Longrightarrow A}{\Delta;\Gamma\Longrightarrow[\Psi]A}\ \Box\mathsf{R} \qquad \frac{\Delta,A[\Psi];\Gamma,[\Psi]A\Longrightarrow C}{\Delta;\Gamma,[\Psi]A\Longrightarrow C}\ \Box\mathsf{L}$$

$$\frac{\Delta;\Gamma\Longrightarrow B_1 \quad \ldots \quad \Delta;\Gamma\Longrightarrow B_m}{\Delta;\Gamma\Longrightarrow B_1,\ldots,B_m}\ \mathsf{ctx}$$

Fig. 2. Sequent Calculus for ICML

These conventions should be understood only as a short-hand notation and do not introduce any essential ambiguity into the system: contexts Γ and Δ should still be considered ordered.

The inference rules for sequents can be derived systematically from the rules for natural deduction. For each connective there are right and left rules, corresponding to the introduction and elimination rules of natural deduction. We have arranged the system so that no explicit rules are required for weakening and contraction, which simplifies the proof of the admissibility of cut and the relationship to natural deduction. We forego the straightforward development and just summarize the resulting system in Figure 2.

These rules are designed to correspond most directly to natural deduction, which means that some of the premises are redundant with respect to provability. Specifically, $A\to B$ in the right premise of ${\to}\mathsf{L}$ and $[\Psi]A$ in the premise of $\Box\mathsf{L}$ are not needed if we are only concerned with provability, although we do need them if we want to model all normal natural deductions directly.

First, we show that cut is admissible in this system. Recall that we follow the general conventions of the sequent calculus and freely allow exchange between assumptions.

Theorem 3.1 Admissibility of Cut.

(1) ***(Weakening)*** *If* $\Delta;\Gamma\Longrightarrow C$ *then* $\Delta,A[\Psi];\Gamma\Longrightarrow C$ *and* $\Delta;\Gamma,A\Longrightarrow C$
(2) ***(Identity)*** $\Delta;\Gamma,A\Longrightarrow A$ *for any proposition* A.
(3) ***(Contextual Identity)*** $\Delta;\Psi\Longrightarrow\Psi$ *for any context* Ψ.
(4) ***(Cut)*** *If* $\Delta;\Gamma\Longrightarrow A$ *and* $\Delta;\Gamma,A\Longrightarrow C$ *then* $\Delta;\Gamma\Longrightarrow C$
(5) ***(Contextual Cut)*** *If* $\Delta;\Psi\Longrightarrow A$ *and* $\Delta,A[\Psi];\Gamma\Longrightarrow C$ *then* $\Delta;\Gamma\Longrightarrow C$

Proof. Weakening is a trivial induction over the given derivation. Note that the size or structure of a derivation does not change when weakening is applied. We can therefore use it tacitly in the proof of the other properties.

The identity principles are proved by induction on the structure of A and Ψ, respectively.

Admissibility of cut and contextual cut is proved by a nested induction, first on the structure of A and $A[\Psi]$, and second on the structure of the two derivations

that are cut. □

Now it is easy to establish that every derivation has a verification and vice versa. We only state the main property along these lines. We commit a slight abuse of notation by writing Γ and Δ on both sides, even though the actual judgments in hypotheses for natural deductions and sequents are different (A true vs. A hyp).

THEOREM 3.2 PROOFS AND VERIFICATIONS. $\Delta;\Gamma \vdash A$ true *iff* $\Delta;\Gamma \Longrightarrow A$.

PROOF. From left to right by induction over the structure of the given proof, using admissibility of identity for hypotheses and cut for elimination rules.

From right to left by induction over the structure of the given verification, using substitution properties. □

By examining the relationships between proofs and verifications we can see that the admissibility of cut is the global version of local soundness, and that the admissibility of identity is the global version of local completeness. A more formal analysis of this relationship is beyond the scope of the present paper.

3.2 Examples

We revisit only two of the examples from Section 2.5, giving the sequent derivation of (5) and showing that (9) is not provable. We use the admissible Identity instead of the init rule to emphasize that A need not be atomic.

$$\dfrac{\dfrac{\dfrac{}{A[\cdot];[\cdot]A \Longrightarrow \cdot}\,\mathsf{ctx} \qquad \dfrac{}{A[\cdot];[\cdot]A, A \Longrightarrow A}\,\mathsf{Identity}}{A[\cdot];[\cdot]A \Longrightarrow A}\,\mathsf{reflect}}{\dfrac{[\cdot]A \Longrightarrow A}{\Longrightarrow [\cdot]A \to A}\,{\to}R}\,{\Box}L$$

We can now show that (9) is not derivable for an arbitrary A. In each sequent there is at most one applicable rule in the bottom-up construction of a potential derivation, leaving us with the subgoal of verifying A in an empty context of hypotheses. This subgoal does not have a verification in general.

$$\dfrac{\dfrac{\cdot \Longrightarrow A}{A \Longrightarrow [\cdot]A}}{\Longrightarrow A \to [\cdot]A}$$

4. SIMPLE CONTEXTUAL MODAL TYPE THEORY

In this section we develop a type theory based on contextual modal logic. We postpone the treatment of dependencies to the next section and cover functions and the contextual modality we have introduced. This could serve as the foundation for a functional language, although there are some difficulties of extending it to a fully dependent type theory, as we see in Section 5.

$$\frac{}{\Delta;(\Gamma,x{:}A,\Gamma')\vdash x:A}\ \mathsf{hyp} \qquad \frac{(\Delta,u{::}A[\Psi],\Delta');\Gamma\vdash\sigma:\Psi}{(\Delta,u{::}A[\Psi],\Delta');\Gamma\vdash \mathsf{clo}(u,\sigma):A}\ \mathsf{ctxhyp}$$

$$\frac{\Delta;\Gamma,x{:}A\vdash M:B}{\Delta;\Gamma\vdash \mathsf{lam}(x{:}A.\,M):A\to B}\ {\to}\mathsf{I} \qquad \frac{\Delta;\Gamma\vdash M:A\to B \quad \Delta;\Gamma\vdash N:A}{\Delta;\Gamma\vdash \mathsf{app}(M,N):B}\ {\to}\mathsf{E}$$

$$\frac{\Delta;\Psi\vdash M:A}{\Delta;\Gamma\vdash \mathsf{box}(\Psi.\,M):[\Psi]A}\ \Box\mathsf{I} \qquad \frac{\Delta;\Gamma\vdash M:[\Psi]A \quad \Delta,u{::}A[\Psi];\Gamma\vdash N:C}{\Delta;\Gamma\vdash \mathsf{letbox}(M,u.\,N):C}\ \Box\mathsf{E}$$

$$\frac{\Delta;\Gamma\vdash N_1:B_1 \quad \ldots \quad \Delta;\Gamma\vdash N_m:B_m}{\Delta;\Gamma\vdash (N_1/y_1,\ldots,N_m/y_m):(y_1{:}B_1,\ldots,y_m{:}B_m)}$$

Fig. 3. Proof Term Assignment for ICML

4.1 Proof Term Assignment

In the terminology of Martin-Löf [1996], the truth judgment for ICML is *synthetic* in that we have to provide a separate proof as evidence for the judgment. By annotating the judgment with proof terms we obtain two benefits: first, the judgment is now *analytic* in the sense that it contains its evidence, and second, it provides us with a notation for programs under the Curry-Howard interpretation of proofs as programs.

When assigning proof terms we have to be careful to respect variable scope and allow consistent renaming of bound variables, because some operations are applied to full contexts and hence to all variables in a term simultaneously. We therefore chose an abstract syntax where binding is made explicit, using a dot (".") to separate a variable or context on the left from its scope on the right.

This proof term assignment uses the following proof term language, where x stands for ordinary variables and u for contextual modal variables. Keeping in mind one of our applications, we will refer to u as standing for *meta-variables*.

$$\begin{array}{rrcl}
\text{Terms} & M,N & ::= & x \mid \mathsf{lam}(x{:}A.\,M) \mid \mathsf{app}(M,N) \\
 & & & \mid \mathsf{clo}(u,\sigma) \mid \mathsf{box}(\Psi.\,M) \mid \mathsf{letbox}(M,u.\,N) \\
\text{Substitutions} & \sigma,\tau & ::= & \cdot \mid \sigma, M/x \\
\text{Contexts} & \Gamma,\Psi & ::= & \cdot \mid \Gamma, x{:}A \\
\text{Modal Contexts} & \Delta & ::= & \cdot \mid \Delta, u{::}A[\Psi]
\end{array}$$

Recall that all variables declared in a context or modal context must be distinct.

Since the binding structure is somewhat non-standard, we explicitly define the

set of free variables for terms, $\mathsf{FV}(M)$, and substitutions, $\mathsf{FV}(\sigma)$.

$$\begin{array}{lcl}
\mathsf{FV}(x) & = & \{x\} \\
\mathsf{FV}(\mathsf{lam}(x{:}A.\, M)) & = & \mathsf{FV}(M) - \{x\} \\
\mathsf{FV}(\mathsf{app}(M, N)) & = & \mathsf{FV}(M) \cup \mathsf{FV}(N) \\
\mathsf{FV}(\mathsf{clo}(u, \sigma)) & = & \mathsf{FV}(\sigma) \\
\mathsf{FV}(\mathsf{box}(\Psi.\, M)) & = & \{\,\} \\
\mathsf{FV}(\mathsf{letbox}(M, u.\, N)) & = & \mathsf{FV}(M) \cup \mathsf{FV}(N) \\
\mathsf{FV}(\cdot) & = & \{\,\} \\
\mathsf{FV}(\sigma, M/x) & = & \mathsf{FV}(\sigma) \cup \mathsf{FV}(M)
\end{array}$$

The term constructor $\mathsf{box}(\Psi.\, M)$ binds all the variables from the context Ψ simultaneously. Consequently, the renaming of bound variables or α-conversion now also includes terms of the form $\mathsf{box}(\Psi.\, M)$ where variables in Ψ may be renamed, and we tacitly apply α-conversion as necessary. Furthermore, according to the typing rule $\Box\mathsf{I}$ from Figure 3, Ψ is required to contain all the free variables of M in $\mathsf{box}(\Psi.\, M)$. Thus, $\mathsf{FV}(\mathsf{box}(\Psi.\, M))$ is appropriately defined to be empty. M may still contain free meta-variables from the modal context Δ, so we next define the free meta-variables in a term and substitution, $\mathsf{FMV}(M)$ and $\mathsf{FMV}(\sigma)$.

$$\begin{array}{lcl}
\mathsf{FMV}(x) & = & \{\,\} \\
\mathsf{FMV}(\mathsf{lam}(x{:}A.\, M)) & = & \mathsf{FMV}(M) \\
\mathsf{FMV}(\mathsf{app}(M, N)) & = & \mathsf{FMV}(M) \cup \mathsf{FMV}(N) \\
\mathsf{FMV}(\mathsf{clo}(u, \sigma)) & = & \{u\} \cup \mathsf{FMV}(\sigma) \\
\mathsf{FMV}(\mathsf{box}(\Psi.\, M)) & = & \mathsf{FMV}(M) \\
\mathsf{FMV}(\mathsf{letbox}(M, u.\, N)) & = & \mathsf{FMV}(M) \cup (\mathsf{FMV}(N) - \{u\}) \\
\mathsf{FMV}(\cdot) & = & \{\,\} \\
\mathsf{FMV}(\sigma, M/x) & = & \mathsf{FMV}(\sigma) \cup \mathsf{FMV}(M)
\end{array}$$

The context Ψ in $\mathsf{box}(\Psi.\, M)$ binds ordinary variables, but does not bind any meta-variables, so the free meta-variables of $\mathsf{box}(\Psi.\, M)$ are those of M. On the other hand, the free meta-variables of $\mathsf{letbox}(M, u.\, N)$ include those of M and N, but u has to be excluded because it is bound in $\mathsf{letbox}(M, u.\, N)$ with the scope extending through N. Of course, bound meta-variables are subject to α-conversion, too.

4.2 Examples

In this section we present the ICML proof terms corresponding to the propositions from the example in Section 2.5. For brevity, we omit the types from the proof terms.

$$M_1 \;:\; [x{:}C]A \to [y_1{:}C, y_2{:}D]A = \\ \mathsf{lam}(z.\, \mathsf{letbox}(z, u.\, \mathsf{box}(y_1, y_2.\, \mathsf{clo}(u, [y_1/x]))))$$

$$M_2 \;:\; [x_1{:}C, x_2{:}C]A \to [y{:}C]A = \\ \mathsf{lam}(z.\, \mathsf{letbox}(z, u.\, \mathsf{box}(y.\, \mathsf{clo}(u, [y/x_1, y/x_2]))))$$

$$M_3 \;:\; [x{:}A]A = \\ \mathsf{box}(x.\, x)$$

$$M_4 \;:\; [x{:}A]B \to [y_1{:}A][y_2{:}B]C \to [z{:}A]C = \\ \mathsf{lam}(x_1.\, \mathsf{lam}(x_2. \\ \quad \mathsf{letbox}(x_1, u.\, \mathsf{letbox}(x_2, v.\, \mathsf{box}(z.\, \mathsf{letbox}(\mathsf{clo}(v, [z/y_1]), \\ \qquad\qquad w.\, \mathsf{clo}(w, [\mathsf{clo}(u, [z/x])/y_2])))))))$$

$$M_5 \;:\; [\cdot]A \to A = \\ \mathsf{lam}(x.\, \mathsf{letbox}(x, u.\, \mathsf{clo}(u, [\cdot])))$$

$$M_6 \;:\; [x{:}C]A \to [y_1{:}D][y_2{:}C]A = \\ \mathsf{lam}(x_1.\, \mathsf{letbox}(x_1, u.\, \mathsf{box}(y_1.\, \mathsf{box}(y_2.\, \mathsf{clo}(u, [y_2/x])))))$$

$$M_7 \;:\; [x{:}C](A \to B) \to [y{:}D]A \to [z_1{:}C, z_2{:}D]B = \\ \mathsf{lam}(x_1.\, \mathsf{lam}(x_2. \\ \quad \mathsf{letbox}(x_1, u.\, \mathsf{letbox}(x_2, v.\, \mathsf{box}(z_1, z_2.\, \mathsf{app}(\mathsf{clo}(u, [z_1/x]), \mathsf{clo}(v, [z_2/y])))))))$$

$$M_8 \;:\; [x{:}A](A \to B) \to [y{:}B]C \to [z{:}A]C = \\ \mathsf{lam}(x_1.\, \mathsf{lam}(x_2. \\ \quad \mathsf{letbox}(x_1, u.\, \mathsf{letbox}(x_2, v.\, \mathsf{box}(z{:}A.\, \mathsf{clo}(v, [\mathsf{app}(\mathsf{clo}(u, [z/x]), z)/y])))))$$

4.3 Substitution on Terms

In this section we define the operations of substitution, simultaneous substitution, and substitution for meta-variables, both into a term and substitution. Substitution for ordinary and meta-variables will be total operations since any side condition can be satisfied by α-conversion. First, ordinary capture-avoiding substitution on a single variable, $[M/x]N$ and $[M/x]\sigma$.

$$\begin{array}{lcl}
[M/x](x) & = & M \\
[M/x](y) & = & y \quad \text{if } y \neq x \\
[M/x](\mathsf{lam}(y{:}B.\, N)) & = & \mathsf{lam}(y{:}B.\, [M/x]N) \quad \text{provided } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
[M/x](\mathsf{app}(N_1, N_2)) & = & \mathsf{app}([M/x]N_1, [M/x]N_2) \\
[M/x](\mathsf{clo}(u, \sigma)) & = & \mathsf{clo}(u, [M/x]\sigma) \\
[M/x](\mathsf{box}(\Psi.\, N)) & = & \mathsf{box}(\Psi.\, N) \\
[M/x](\mathsf{letbox}(N_1, u.\, N_2)) & = & \mathsf{letbox}([M/x]N_1, u.\, [M/x]N_2) \\
[M/x](\cdot) & = & \cdot \\
[M/x](\sigma, N/y) & = & [M/x]\sigma, ([M/x]N)/y
\end{array}$$

The substitution commutes with the constructors in all the cases, except in the case for $\mathsf{box}(\Psi.\, M)$. As already discussed, $\mathsf{box}(\Psi.\, M)$ does not contain any free variables, and in particular, it does not contain the free variable x. Thus, $\mathsf{box}(\Psi.\, M)$ is not changed when $[M/x]$ is applied to it.

This substitution satisfies the following versions of the substitution principle, annotated with proof terms.

Theorem 4.1 Substitution on Terms.

(1) If $\Delta;\Gamma \vdash M : A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash N : C$ *then* $\Delta;\Gamma,\Gamma' \vdash [M/x]N : C$.

(2) If $\Delta;\Gamma \vdash M : A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash \sigma : \Psi$ *then* $\Delta;\Gamma,\Gamma' \vdash [M/x]\sigma : \Psi$.

Proof. By induction on the structure of the second given derivation. □

Next we define simultaneous substitution $[\sigma]M$ and $[\sigma]\tau$. It is only total when the substitution σ is defined on all free variables in M and τ, respectively. This will be satisfied, because simultaneous substitution is only applied when the assumptions of the theorem following this definition are satisfied. Simultaneous substitutions commute with the term constructors, as one would expect. The only exception are the terms of the form $\mathsf{box}(\Psi.\, M)$; they do not contain any free variables and thus remain unchanged by the substitution.

$$
\begin{array}{lcl}
[\sigma_1, M/x, \sigma_2](x) & = & M \\
[\sigma](\mathsf{lam}(y{:}B.\, N)) & = & \mathsf{lam}(y{:}B.\, [\sigma, y/y]N) \quad \text{if } y \notin \mathsf{FV}(\sigma) \text{ and } y \notin \mathsf{dom}(\sigma) \\
[\sigma](\mathsf{app}(N_1, N_2)) & = & \mathsf{app}([\sigma]N_1, [\sigma]N_2) \\
[\sigma](\mathsf{clo}(u, \tau)) & = & \mathsf{clo}(u, [\sigma]\tau) \\
[\sigma](\mathsf{box}(\Psi.\, N)) & = & \mathsf{box}(\Psi.\, N) \\
[\sigma](\mathsf{letbox}(N_1, u.\, N_2)) & = & \mathsf{letbox}([\sigma]N_1, u.\, [\sigma]N_2) \\
[\sigma](\cdot) & = & \cdot \\
[\sigma](\tau, N/y) & = & [\sigma]\tau, ([\sigma]N)/y
\end{array}
$$

Sometimes we need to rename the domain of a substitution to match a given context. When $\sigma = (M_1/x_1, \ldots, M_n/x_n)$ and $\Psi = (y_1{:}A_1, \ldots, y_n{:}A_n)$ then $\sigma/\Psi = (M_1/y_1, \ldots, M_n/y_n)$.

Simultaneous substitutions satisfy the simultaneous substitution principle, annotated with proof terms. The second property amounts to composition of the substitutions τ and σ.

Theorem 4.2 Simultaneous Substitution on Terms.

(1) If $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Psi \vdash N : C$ *then* $\Delta;\Gamma \vdash [\sigma]N : C$.

(2) If $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Psi \vdash \tau : \Theta$ *then* $\Delta;\Gamma \vdash [\sigma]\tau : \Theta$.

Proof. By induction on the structure of the second given derivation. □

Substitutions for meta-variables u are a little more difficult. One complication is that simultaneous substitutions σ are not first class: from the logical point of view it makes sense to restrict them to $\mathsf{clo}(u, \sigma)$. So we cannot just substitute a term M to obtain $\mathsf{clo}(M, \sigma)$ because the latter is not well-formed. The solution to this problem becomes clear when we recall that u should stand for meta-variables. The

closure $\mathsf{clo}(u, \sigma)$ is a postponed substitution. As soon as we know which term u should stand for we can apply σ to it.

Moreover, because of α-conversion, the variables that are substituted at different occurrences of u may be different. This is because a declaration such as $u{::}A[x{:}B]$ is indistinguishable from $u{::}A[x'{:}B]$, by renaming. This is a bit pedantic here, but unavoidable in the dependent case. As a result, substitution for a meta-variable must carry a context, written as $[\![\Psi.M/u]\!]N$ and $[\![\Psi.M/u]\!]\sigma$ where Ψ binds all free variables in M. This complication can be eliminated in an implementation of our calculus based on de Bruijn indexes.

$$
\begin{array}{lcl}
[\![\Psi.M/u]\!](x) & = & x \\
[\![\Psi.M/u]\!](\mathsf{lam}(y{:}B.\, N)) & = & \mathsf{lam}(y{:}B.\, [\![\Psi.M/u]\!]N) \\
[\![\Psi.M/u]\!](\mathsf{app}(N_1, N_2)) & = & \mathsf{app}([\![\Psi.M/u]\!]N_1, [\![\Psi.M/u]\!]N_2) \\
[\![\Psi.M/u]\!](\mathsf{clo}(u, \tau)) & = & [[\![\Psi.M/u]\!]\tau/\Psi]M \\
[\![\Psi.M/u]\!](\mathsf{clo}(v, \tau)) & = & \mathsf{clo}(v, [\![\Psi.M/u]\!]\tau) \quad \text{provided } v \neq u \\
[\![\Psi.M/u]\!](\mathsf{box}(\Gamma.\, N)) & = & \mathsf{box}(\Gamma.\, [\![\Psi.M/u]\!]N) \\
[\![\Psi.M/u]\!](\mathsf{letbox}(N_1, v.\, N_2)) & = & \mathsf{letbox}([\![\Psi.M/u]\!]N_1, v.\, [\![\Psi.M/u]\!]N_2) \\
 & & \quad \text{provided } v \notin \mathsf{FMV}(M) \text{ and } v \neq u \\
[\![\Psi.M/u]\!](\cdot) & = & \cdot \\
[\![\Psi.M/u]\!](\tau, N/y) & = & [\![\Psi.M/u]\!]\tau, ([\![\Psi.M/u]\!]N)/y
\end{array}
$$

Applying $[\![\Psi.M/u]\!]$ to the closure $\mathsf{clo}(u, \tau)$ first obtains the simultaneous substitution $\tau' = [\![\Psi.M/u]\!]\tau$, but instead of returning $\mathsf{clo}(M, \tau')$, it proceeds to eagerly apply τ' to M. Before τ' can be carried out, however, it's domain must be renamed to match the variables in Ψ, denoted by τ'/Ψ.

While the definition of the discussed case may seem circular at first, it is actually well-founded. The computation of τ' recursively invokes $[\![\Psi.M/u]\!]$ on τ, a constituent of $\mathsf{clo}(u, \tau)$. Then τ'/Ψ is applied to M, but applying simultaneous substitutions has already been defined without appeal to meta-variable substitution.

As an illustration of this operation, consider the expression

$$[\![x{:}A \to A, y{:}A.\mathsf{app}(x, y)/u]\!]\mathsf{clo}(u, [\mathsf{lam}(z{:}A.\, z)/x, t/y])$$

Here we assume that $u{::}A[x{:}A \to A, y{:}A]$ and $t{:}A$ (and t does not depend on u), so that all the terms involved are well typed. We obtain $\mathsf{app}(\mathsf{lam}(z{:}A.\, z), t)$, because upon substituting $\mathsf{app}(x, y)$ for u, the local variables x and y are immediately replaced with $\mathsf{lam}(z{:}A.\, z)$ and t, respectively.

Another property worth emphasizing here is that the side conditions in the definition of meta-variable substitution only involve meta-variables, but no checks involving ordinary variables are needed. This property will be exploited in Section 7 to efficiently implement meta-variables in a logical framework.

Substitution of a meta-variable satisfies the following contextual substitution property. Again, this is the contextual substitution property from Section 2.6 annotated with proof terms.

Theorem 4.3 Contextual Substitution on Terms.

(1) *If* $\Delta; \Psi \vdash M : A$ *and* $(\Delta, u{::}A[\Psi], \Delta'); \Gamma \vdash N : C$
then $(\Delta, \Delta'); \Gamma \vdash [\![\Psi.M/u]\!]N : C$.

(2) *If* $\Delta; \Psi \vdash M : A$ *and* $(\Delta, u{::}A[\Psi], \Delta'); \Gamma \vdash \tau : \Theta$
then $(\Delta, \Delta'); \Gamma \vdash [\![\Psi.M/u]\!]\tau : \Theta$.

PROOF. By simple inductions on the second given derivation, appealing to Theorem 4.2 in the case for meta-variables. □

4.4 Proof Reductions and Expansions

In this section we present the local reductions and expansions, employing the substitution operations defined above. First, the two local reductions.

$$\begin{array}{lcl} \mathsf{app}(\mathsf{lam}(x{:}A.\, N), M) & \Longrightarrow_R & [M/x]N \\ \mathsf{letbox}(\mathsf{box}(\Psi.\, M), u.\, N) & \Longrightarrow_R & [\![\Psi.M/u]\!]N \end{array}$$

In order to formulate the local expansion for the contextual modality, we need to construct identity substitutions. The corresponding operation id, is defined for each context Ψ as follows.

$$\begin{array}{lcl} \mathsf{id}_{(\cdot)} & = & (\cdot) \\ \mathsf{id}_{(\Psi, x:A)} & = & (\mathsf{id}_\Psi, x/x) \end{array}$$

Obviously, id_Ψ stands for different substitutions, depending on the particular instance of Ψ. In other words, id is a meta-level operation just like substitution, and not a proper part of the proof term calculus. We easily verify that it is indeed an identity substitution.

THEOREM 4.4 IDENTITY SUBSTITUTION.

(1) $\Delta; \Psi \vdash \mathsf{id}_\Psi : \Psi$ *for any* Ψ.
(2) $[\mathsf{id}_\Psi]M = M$ *provided* $\Delta; \Psi \vdash M : A$ *for some* Δ *and* A.
(3) $[\mathsf{id}_\Psi]\sigma = \sigma$ *provided* $\Delta; \Psi \vdash \sigma : \Gamma$ *for some* Δ *and* Γ.

PROOF. By easy inductions. □

Now we can write out the two local expansions.

$$\begin{array}{lcll} M : A \to B & \Longrightarrow_E & \mathsf{lam}(x{:}A.\, \mathsf{app}(M, x)) & \text{for } x \notin \mathsf{FV}(M) \\ M : [\Psi]A & \Longrightarrow_E & \mathsf{letbox}(M, u.\, \mathsf{box}(\Psi.\, \mathsf{clo}(u, \mathsf{id}_\Psi))) & \end{array}$$

Despite appearances, the second rule shows, as expected, that we can always create a term of type $[\Psi]A$ by introduction (the inner box). The fact that the elimination is a let form makes it textually appear on the outside, but conceptually we apply an elimination to M, name the result u, and then reconstruct $\mathsf{box}(\Psi.\, \mathsf{clo}(u, \mathsf{id}_\Psi))$ of type $[\Psi]A$ from u.

Due to the presence of the letbox elimination, the calculus also requires two commuting reductions to reach a normal form in which no types are introduced and later eliminated. An example to illustrate the need for the commuting reduction is

$$\begin{array}{l} \mathsf{lam}(x{:}[\cdot]A.\, \mathsf{app}(\mathsf{letbox}(x, u.\, \mathsf{lam}(z{:}[\cdot]A \to [\cdot]A.\, \mathsf{app}(z, \mathsf{box}(\cdot.\, \mathsf{clo}(u, \cdot))))), \\ \qquad\qquad \mathsf{lam}(w{:}[\cdot]A.\, w))) \end{array}$$

In a more natural mathematical syntax:

$$\lambda x.(\mathsf{let}\ \mathsf{box}\, u = x\ \mathsf{in}\ \lambda z.z\ (\mathsf{box}\ u)\ \mathsf{end})\ (\lambda w.w)$$

This term has no redex because the letbox hides the application of $\lambda z.z\ (\mathsf{box}\ u)$ to $\lambda w.w$. This leads to the following commuting reductions:

$$\begin{array}{lll}
\mathsf{app}(\mathsf{letbox}(N_1, v.\, N_2), M) & \Longrightarrow_C & \mathsf{letbox}(N_1, v.\, \mathsf{app}(N_2, M)) \\
& & \quad \text{provided } v \notin \mathsf{FMV}(M) \\
\mathsf{letbox}(\mathsf{letbox}(N_1, v.\, N_2), u.\, M) & \Longrightarrow_C & \mathsf{letbox}(N_1, v.\, \mathsf{letbox}(N_2, u.\, M)) \\
& & \quad \text{provided } v \notin \mathsf{FMV}(u.M)
\end{array}$$

It is easy to see from the substitution properties that local reductions, local expansions, and commuting reductions preserve types. In this proof, and also later in the paper, we use *inversion* in a purely syntactic manner: when we know a judgment has a derivation, and the judgment matches the conclusion of only one rule, then the premises of that rule must also have derivations. A slight generalization allows a finite number of rules to match the judgment, in which case we have to split the proof into several cases.

Theorem 4.5 Subject Reduction and Expansion.

(1) *If* $\Delta; \Gamma \vdash M : A$ *and* $M \Longrightarrow_R M'$ *then* $\Delta; \Gamma \vdash M' : A$.

(2) *If* $\Delta; \Gamma \vdash M : A$ *and* $M : A \Longrightarrow_E M'$ *then* $\Delta; \Gamma \vdash M' : A$.

(3) *If* $\Delta; \Gamma \vdash M : A$ *and* $M \Longrightarrow_C M'$ *then* $\Delta; \Gamma \vdash M' : A$.

Proof. For subject reduction by inversion on the given typing derivation and appeal to one of the substitution properties. For subject expansion the proof is direct. For commuting reductions by inversion on the giving typing derivation and weakening. □

4.5 Strong Normalization

In order to prove strong normalization, we can exploit strong normalization for the simply-typed λ-calculus with sums under permutation conversions [de Groote 2002]. The result is somewhat weaker than the corresponding property for the usual calculi of explicit substitution, because we treat application of substitutions as a single-step meta-level operation. For a reduction relation $\Longrightarrow$, we write $\Longrightarrow^*$ for zero or more reductions and $\Longrightarrow^+$ for one or more reductions. We omit the straightforward congruence rules for all reductions and note that subject reduction (Theorem 4.5) continues to hold.

For the target of our translation, we use the following fragment of the simply-typed λ-calculus, where a stands for base types.

$$\begin{array}{lll}
\text{Types} & \alpha, \beta, \gamma & ::= a \mid \alpha \to \beta \mid \alpha + \beta \\
\text{Terms} & s, t, r & ::= x \mid \lambda x.t \mid s\, t \mid \iota_1(t) \mid \iota_2(t) \mid \delta(s, x.t, y.r)
\end{array}$$

λ-abstraction and application are as usual, ι_1 and ι_2 are left and right injections into a sum, and δ represents a case construct. We continue to use our convention that $x.t$ binds x with scope t. We omit the standard typing rules but show the reductions. De Groote calls the last two *permutation conversions* even though they

are applied as reductions.

$$
\begin{array}{lll}
(\lambda x.t)\, s & \Longrightarrow_D & [s/x]t \\
\delta(\iota_1(s), x.t, y.r) & \Longrightarrow_D & [s/x]t \\
\delta(\iota_2(s), x.t, y.r) & \Longrightarrow_D & [s/y]r \\
\delta(s, x.t, y.r)\, p & \Longrightarrow_D & \delta(s, x.t\, p, y.r\, p) \quad \text{provided } x \notin \mathsf{FV}(p),\ y \notin \mathsf{FV}(p) \\
\delta(\delta(s, x.t, y.r), u.p, v.q) & \Longrightarrow_D & \delta(s, x.\delta(t, u.p, v.q), y.\delta(r, u.p, v.q)) \\
 & & \quad \text{provided } x \notin \mathsf{FV}(u.p, v.q),\ y \notin \mathsf{FV}(u.p, v.q)
\end{array}
$$

We use three auxiliary operations on terms and types in the target language to form iterated function types, abstract over a context, and apply a term to a sequence of arguments.

$$
\begin{array}{rcl}
\Pi(x_1{:}\beta_1, \ldots, x_m{:}\beta_m)(\alpha) & = & \beta_1 \to \cdots \to \beta_m \to \alpha \\
\Lambda(x_1{:}\beta_1, \ldots, x_m{:}\beta_m)(s) & = & \lambda x_1 \ldots \lambda x_m.s \\
\mathsf{App}(s)(t_1, \ldots, t_m) & = & s\, t_1 \ldots\, t_m
\end{array}
$$

The interpretation of types is a simple compositional translation $(A)^o$, where $[y_1{:}B_1, \ldots, y_m{:}B_m]A$ is mapped $\alpha + \alpha$, where $\alpha = B_1^o \to \cdots B_m^o \to A^o$. The sum is necessary because the target of the translation does not have a **let** construct. We also extend this translation to ordinary hypotheses and contextual validity assumptions.

$$
\begin{array}{lcl}
(a)^o & = & a \\
(A \to B)^o & = & A^o \to B^o \\
([\Psi]A)^o & = & (\Pi(\Psi^o)(A^o)) + (\Pi(\Psi^o)(A^o)) \\
(\cdot)^o & = & \cdot \\
(\Gamma, x{:}A)^o & = & \Gamma^o, x{:}A^o \\
(\cdot)^o & = & \cdot \\
(\Delta, u{::}A[\Psi])^o & = & \Delta^o, u{:}\Pi(\Psi^o)(A^o)
\end{array}
$$

In the translation of terms $(M)^o$, meta-variables are mapped to ordinary variables that take additional arguments, and closures $\mathsf{clo}(u, (M_1/y_1, \ldots, M_m/y_m))$ are mapped to corresponding sequences of applications $((u\, M_1) \ldots M_m)$. The letbox construct is translated to a case construct in order to preserve the commuting reductions.

$$
\begin{array}{rcl}
(x)^o & = & x \\
(\mathsf{lam}(x{:}A.\, M))^o & = & \lambda x.M^o \\
(\mathsf{app}(M, N))^o & = & M^o\, N^o \\
(\mathsf{clo}(u, \sigma))^o & = & \mathsf{App}(u)(\sigma^o) \\
(\mathsf{box}(\Psi.\, M))^o & = & \iota_1(\Lambda(\Psi^o)(M^o)) \\
(\mathsf{letbox}(M, u.\, N))^o & = & \delta(M^o, u.N^o, u.N^o) \\
(\cdot)^o & = & \cdot \\
(\sigma, M/x)^o & = & \sigma^o, M^o
\end{array}
$$

The choice of the first or second injection in the translation of box is arbitrary.

Under these translations, the following theorem is easy to show. We omit some properties pertaining to simultaneous substitutions that follow directly from the properties of terms.

THEOREM 4.6 INTERPRETATION.

(1) *If* $\Delta;\Gamma \vdash M : A$ *then* $\Delta^o, \Gamma^o \vdash M^o : A^o$.
(2) *If* $\Delta;\Gamma \vdash M : A$ *and* $\Delta;\Gamma, x{:}A, \Gamma' \vdash N : B$ *then* $([M/x]N)^o = [M^o/x]N^o$.
(3) *If* $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Psi \vdash M : A$ *then* $\mathsf{App}(\Lambda(\Psi^o)(M^o))(\sigma^o) \Longrightarrow_D^* ([\sigma]M)^o$.
(4) *If* $\Delta;\Psi \vdash M : A$ *and* $(\Delta, u{::}A[\Psi], \Delta');\Gamma \vdash N : C$ *then* $[\Lambda(\Psi^o)(M^o)/u]N^o \Longrightarrow_D^* ([\![\Psi.M/u]\!]N)^o$.

PROOF. By simple structural inductions. The typing assumptions are necessary for the substitution properties because $[M/x](\mathsf{box}(\Psi.\, N)) = \mathsf{box}(\Psi.\, N)$. The reductions are necessary in the last part to eliminate potential redexes after substituting a λ-abstraction for u. □

THEOREM 4.7 STRONG NORMALIZATION.

(1) *If* $\Delta;\Gamma \vdash M : A$ *and* $M \Longrightarrow_R N$ *or* $M \Longrightarrow_C N$ *then* $M^o \Longrightarrow_D^+ N^o$.
(2) *The combination of* $\Longrightarrow_R$ *and* $\Longrightarrow_C$ *is strongly normalizing on well-typed terms.*

PROOF. Part (1) follows by definition of $\Longrightarrow_R$ and $\Longrightarrow_C$, inversion on the typing derivation, and appeal to the preceding theorem. Part (2) follows from Part (1) and the strong normalization theorem for the simply-typed λ-calculus with sums [de Groote 2002]. □

5. DEPENDENT CONTEXTUAL MODAL TYPE THEORY

As shown in the previous section, a type theory with the contextual modal operator requires commuting reductions and therefore lacks unique canonical forms. In the context of functional programming this is acceptable, because commuting reductions never arise when evaluating a closed term because we do not normalize under variable binding operators. For a logical framework such as LF [Harper et al. 1993], however, object-language expressions and deductions are represented by canonical forms in the type theory underlying the logical framework. Therefore commuting conversions are to be avoided: they complicate the definitional equality and make it difficult to obtain adequate encodings due to the lack of uniqueness of canonical forms.

In this section we define a dependent contextual modal type theory in the LF family. In order to avoid commuting conversions, we internalize contextual validity not via $[\Psi]A$, but instead we internalize the hypothetical validity judgment. In brief, we pass from $u{::}A[\Psi] \vdash M : C$ to $\mathsf{mlam}(u.\, M) : \Pi u{::}A[\Psi].C$ instead of from $\Psi \vdash M : C$ to $\mathsf{box}(\Psi.\, M) : [\Psi]C$ as before. A similar technique has been employed in the definition of linear logical frameworks in order to retain canonical forms [Hodas and Miller 1994; Cervesato and Pfenning 2002]. We exploit here a recent presentation technique for logical frameworks due to Watkins et al. [2002] in which only canonical forms are well-typed. In order to achieve this we divide the term calculus into *atomic objects* R and *normal objects* M. We reuse the same letters as before for various syntactic categories; the reader should therefore keep in mind that they are appropriately restricted in this section. In particular, all references to M and N stand for normal objects, and all references to A, B, or C stand for normal types. Furthermore, contexts Γ and Ψ contain only declarations $x{:}A$ where A is normal, all terms occurring in substitutions σ are either normal (in N/x) or atomic

(in $R/\!\!/x$), and so on. Finally, while the syntax only guarantees that terms N are normal (that is, contain no β-redexes), the typing rules will in addition guarantee that all well-typed terms are fully η-expanded.

$$
\begin{array}{rrcl}
\text{Normal Kinds} & K & ::= & \mathsf{type} \mid \Pi x{:}A.K \mid \Pi u{::}A[\Psi].K \\
\text{Atomic Types} & P, Q & ::= & a \mid \mathsf{app}(P, N) \mid \mathsf{mapp}(P, \hat{\Psi}.N) \\
\text{Normal Types} & A, B, C & ::= & P \mid \Pi x{:}A.B \mid \Pi u{::}A[\Psi].B \\
\text{Atomic Objects} & R & ::= & x \mid \mathsf{app}(R, N) \mid \mathsf{clo}(u, \sigma) \mid \mathsf{mapp}(R, \hat{\Psi}.N) \\
\text{Normal Objects} & M, N & ::= & \mathsf{lam}(x.\, M) \mid \mathsf{mlam}(u.\, M) \mid R \\
\text{Contexts} & \Gamma, \Psi & ::= & \cdot \mid \Gamma, x{:}A \\
\text{Substitutions} & \sigma & ::= & \cdot \mid \sigma, N/x \mid \sigma, R/\!\!/x \\
\text{Modal Contexts} & \Delta & ::= & \cdot \mid \Delta, u{::}A[\Psi] \\
\text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A
\end{array}
$$

Since we do not need types inside objects, we write $\hat{\Psi}$ for a list of variables $x_1, \ldots, x_n$ which we think of as a context Ψ without types.

Signatures declare global constants and never change in the course of a typing derivation. We therefore suppress the signatures throughout. The modal restriction on occurrences of ordinary variables does not apply to constants, since they are assumed to be meaningful in all possible worlds. This can be seen specifically in the $\Pi^{\Box}\mathsf{E}$ rule, where Γ is not available in the second premise, but the (implicit) signature Σ is propogated unchanged to both premises.

At the heart of the presentation technique are two observations. The first is that we can characterize canonical forms via bi-directional type-checking. The second is that we can formulate normalization as a primitive recursive functional, exploiting the structure of the types and objects. To see the need for the latter, consider the standard rule for application

$$\frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : [N/x]B}\ \Pi\mathsf{E}$$

This is not correct here because $[N/x]B$ is in general not a canonical form. Instead we need to define a form of substitution $[N/x]^a_A(B)$ which is guaranteed to return a canonical form, given that N is canonical at type A, and B is a canonical type. We postpone the somewhat tedious definition of this operator to Section 5.1, where one can also find the related operations $[\![\hat{\Psi}.M/u]\!]^a_{A[\Psi]}(B)$, $[\sigma]^a_{\Psi}(B)$, and auxiliary definitions. The superscript a expresses that the substitution is applied to a type; other superscripts indicate substitution into other kinds of expressions (e.g., n for normal terms or r for atomic terms). We refer to these operations collectively as *hereditary substitutions*.

Typing at the level of objects is divided into three judgments:

$$
\begin{array}{ll}
\Delta; \Gamma \vdash M \Leftarrow A & \text{Check normal object } M \text{ against canonical } A \\
\Delta; \Gamma \vdash R \Rightarrow A & \text{Synthesize canonical } A \text{ for atomic object } R \\
\Delta; \Gamma \vdash \sigma \Leftarrow \Psi & \text{Check } \sigma \text{ against } \Psi
\end{array}
$$

We always assume that Δ and Γ and the subject (M, R, or σ) are given, and that the contexts Δ and Γ contain only canonical types. For checking $M \Leftarrow A$ we also

assume A is given and canonical, and similarly for checking $\sigma \Leftarrow \Psi$ we assume Ψ is given and all types in it are canonical. For synthesis $R \Rightarrow A$ we assume R is given and we generate a canonical A. Similarly, at the level of types we have

$\Delta; \Gamma \vdash A \Leftarrow \mathsf{type}$ Check normal type A
$\Delta; \Gamma \vdash P \Rightarrow K$ Synthesize kind K for atomic type family P

with corresponding assumptions on the constituents. We omit the judgments for well-formed kinds; the ones for contexts are given later in this section.

Judgmental Rules. When checking a normal object that happens to be atomic (that is, has the form R) against a type A we have to synthesize the type for R and compare it with A. Since all synthesized types are canonical, this comparison is simple α-conversion. Moreover, we need to enforce that A is atomic. Otherwise our terms would be β-normal but not necessarily η-long.

$$\frac{\Delta; \Gamma \vdash R \Rightarrow P' \quad P' = P}{\Delta; \Gamma \vdash R \Leftarrow P} \Rightarrow\Leftarrow$$

The rules for ordinary variables and constants are as in the simply typed case. For meta-variables (rule mvar) we need to be careful about directions and dependencies. While $\mathsf{clo}(u, \sigma)$ synthesizes a type, we need the type of u, namely $A[\Psi]$ so we can check σ against Ψ. However, the variables Ψ in $A[\Psi]$ are bound variables with scope A; we assume they are tacitly renamed to match the domain of σ. Moreover, we need to apply σ to transport A from Ψ (upon which it may depend) to Γ. The hereditary simultaneous substitution $[\sigma]^a_\Psi(A)$ always returns a canonical type.

$$\frac{}{\Delta; \Gamma, x{:}A, \Gamma' \vdash x \Rightarrow A}\ \mathsf{var} \qquad \frac{c{:}A \in \Sigma}{\Delta; \Gamma \vdash c \Rightarrow A}\ \mathsf{con}$$

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \mathsf{clo}(u, \sigma) \Rightarrow [\sigma]^a_\Psi(A)}\ \mathsf{mvar}$$

Dependent Functions. Now that our type theory is dependent, we have to be concerned about the well-formedness of types. The formation rule for dependent function types is familiar and straightforward.

$$\frac{\Delta; \Gamma \vdash A \Leftarrow \mathsf{type} \quad \Delta; \Gamma, x{:}A \vdash B \Leftarrow \mathsf{type}}{\Delta; \Gamma \vdash \Pi x{:}A.B \Leftarrow \mathsf{type}}\ \Pi\mathsf{F}$$

This formation rule entails some restrictions on the occurrences of x in B, since x is an ordinary variable rather than a meta-variable. For example, x cannot occur (free) in $\hat{\Psi}.M$, which is enforced in the $\Pi^\Box\mathsf{E}$ rule for $\mathsf{mapp}(R, \hat{\Psi}.M)$, explained later. This is important for the consistency of the type theory.

In general, introduction forms for a type constructor break down a type when read from the conclusion to the premise. This means if the type in the conclusion is given, we can extract the type for the premise, and therefore introduction forms should be checked against a type. Conversely, elimination forms break down the type when read from premise to conclusion. This means if the type in the premise can be synthesized, we can extract the component type for the conclusion, and therefore

elimination forms should synthesize their type. Together with the consideration that the types of variables should always be known, this yields the following rule for the dependent function types $\Pi x{:}A.B$.

$$\frac{\Delta;\Gamma, x{:}A \vdash M \Leftarrow B}{\Delta;\Gamma \vdash \mathsf{lam}(x.\, M) \Leftarrow \Pi x{:}A.B}\ \Pi\mathsf{I} \qquad \frac{\Delta;\Gamma \vdash R \Rightarrow \Pi x{:}A.B \quad \Delta;\Gamma \vdash M \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{app}(R, M) \Rightarrow [M/x]^a_A(B)}\ \Pi\mathsf{E}$$

Contextual Modal Functions. Contextual modal functions can also be dependent, but the corresponding constructor binds a meta-variable instead of an ordinary variable.

$$\frac{\Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type} \quad \Delta, u{::}A[\Psi];\Gamma \vdash B \Leftarrow \mathsf{type}}{\Delta;\Gamma \vdash \Pi u{::}A[\Psi].B \Leftarrow \mathsf{type}}\ \Pi^{\Box}\mathsf{F}$$

The second premise shows that for a declaration $u{::}A[\Psi]$, A must be a valid type in Ψ. In particular, Ψ can not mention any ordinary variables in Γ from the surrounding context. This restriction on the formation of dependent function types is again essential in order to obtain a clean language for meta-variables.

The rule for meta-variables should now be easy to follow, given the foregoing development. The introduction rule for forming a modal function is straightforward. The argument to a modal function must bind its free variables to avoid unexpected capture and related problems. However, the types of these free variables are not needed, since they can be obtained from the type synthesized by R. We therefore write here $\hat{\Psi}$ for the result of erasing types from the context Ψ. Some renaming of bound variables may be necessary to apply this rule to bring the variables in $A[\Psi]$ in accordance with the variables in $\hat{\Psi}.M$.

$$\frac{\Delta, u{::}A[\Psi];\Gamma \vdash M \Leftarrow B}{\Delta;\Gamma \vdash \mathsf{mlam}(u.\, M) \Leftarrow \Pi u{::}A[\Psi].B}\ \Pi^{\Box}\mathsf{I}$$

$$\frac{\Delta;\Gamma \vdash R \Rightarrow \Pi u{::}A[\Psi].B \quad \Delta;\Psi \vdash M \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{mapp}(R, \hat{\Psi}.M) \Rightarrow [\![\hat{\Psi}.M/u]\!]^a_{A[\Psi]}(B)}\ \Pi^{\Box}\mathsf{E}$$

Simultaneous Substitutions. Simultaneous substitutions are checked against a context which prescribes types for the terms in the substitution. The principal complication here are dependencies. When we check $\Delta;\Gamma \vdash (\sigma, M/x) \Leftarrow (\Psi, x{:}A)$ then A will be canonical in Ψ, while the substitution itself is checked in Γ. So we need to apply σ to A before checking M against the result. However, this substitution must return a canonical type, so we once again need hereditary substitution.

$$\frac{}{\Delta;\Gamma \vdash (\cdot) \Leftarrow (\cdot)}$$

$$\frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash M \Leftarrow [\sigma]^a_{\Psi}(A)}{\Delta;\Gamma \vdash (\sigma, M/x) \Leftarrow (\Psi, x{:}A)}$$

$$\frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash R \Rightarrow A' \quad A' = [\sigma]^a_{\Psi}(A)}{\Delta;\Gamma \vdash (\sigma, R/\!\!/x) \Leftarrow (\Psi, x{:}A)}$$

Besides M/x for canonical M, there is a second way to construct a substitution to replace a variable by an atomic term R, written $R/\!\!/x$. This is justified from the nature of hypothetical judgments, since an assumption $x{:}A$ represents $x \Rightarrow A$ so we can substitute R for x if $R \Rightarrow A$. Again, dependencies slightly complicate the form of the rule which is the third one listed above.

Substitutions $R/\!\!/x$ are necessary so that we can extend a given substitution with $x/\!\!/x$ when traversing a binding operator in a type-free way. We could not extend substitutions with x/x, since x is not a canonical term unless it is of atomic type. Identity substitutions can now have the form $x_1/\!\!/x_1, \ldots, x_n/\!\!/x_n$.

Atomic Types. Atomic types mirror the structure of atomic terms, except that the head of an atomic type is always a constant and never a variable.

$$\frac{a{:}K \in \Sigma}{\Delta;\Gamma \vdash a \Rightarrow K}\ \mathsf{acon} \qquad \frac{\Delta;\Gamma \vdash P \Rightarrow K \quad K = \mathsf{type}}{\Delta;\Gamma \vdash P \Leftarrow \mathsf{type}}\ \Rightarrow\Leftarrow$$

$$\frac{\Delta;\Gamma \vdash P \Rightarrow \Pi x{:}A.K \quad \Delta;\Gamma \vdash N \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{app}(P,N) \Rightarrow [N/x]^k_A(K)}\ \Pi\mathsf{E}$$

$$\frac{\Delta;\Gamma \vdash P \Rightarrow \Pi u{::}A[\Psi].K \quad \Delta;\Psi \vdash N \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{mapp}(P,\hat{\Psi}.N) \Rightarrow [\![\hat{\Psi}.N/u]\!]^k_{A[\Psi]}(K)}\ \Pi^{\Box}\mathsf{E}$$

Well-Formed Contexts. The context formation rules are foreshadowed by the formation rules for types. We have the judgments $\Delta \vdash \Gamma\ \mathsf{ctx}$ and $\vdash \Delta\ \mathsf{mctx}$. We omit the straightforward judgments defining the well-formedness of signatures.

$$\frac{}{\Delta \vdash \cdot\ \mathsf{ctx}} \qquad \frac{\Delta \vdash \Gamma\ \mathsf{ctx} \quad \Delta;\Gamma \vdash A \Leftarrow \mathsf{type}}{\Delta \vdash (\Gamma, x{:}A)\ \mathsf{ctx}}$$

$$\frac{}{\vdash \cdot\ \mathsf{mctx}} \qquad \frac{\vdash \Delta\ \mathsf{mctx} \quad \Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type}}{\vdash (\Delta, u{::}A[\Psi])\ \mathsf{mctx}}$$

We show these rules here to emphasize that the context Δ of meta-variables is ordered from left to right: meta-variables declared earlier can appear in the types of later meta-variables, but not vice versa.

5.1 Hereditary Substitution Operations

The substitution functions we need must construct canonical terms, since those are the only ones that are well-formed (and the only ones of interest in a logical framework). Hence, in places where the ordinary substitution would construct a redex $(\lambda y.\, M)\, N$ we must continue, substituting N for y in M. But that could again create a redex, so any redex created by this embedded substitution operation must also continue, and so on. We therefore call this operation *hereditary substitution*.

The difficulty is that hereditary substitution could easily fail to terminate. Consider, for example, $[(\lambda x.\, x\, x)/y]^h(y\, y)$ which arises if one tries to compute the normal form of $(\lambda y.\, y\, y)\, (\lambda x.\, x\, x)$. Clearly, on well-typed terms this should not occur. Moreover, in a typing derivation in our system, hereditary substitutions are only ever applied to terms already checked. Nonetheless, a typing derivation itself contains

appeals to hereditary substitutions, so exploiting (dependent) typing derivations in the development of the theory of hereditary substitutions is not straightforward.

Fortunately, hereditary substitution can be defined as a primitive recursive functional if we pass in the type of the variable we substitute for. For example, if we hereditarily substitute $[\lambda y.M/x](x\,N)$ then, if everything is well-typed, $x{:}A_1 \to A_2$ for some A_1 and A_2. So we would write $[\lambda y.M/x]_{A_1\to A_2}(x\,N)$, indexing the operation with the type of x. If the substitution is to continue hereditarily we must further substitute N for y in M, again hereditarily. But notice that if the original substitution is well-typed then $y{:}A_1$, so we invoke $[N/y]_{A_1}M$. Even though we switched from substituting into $x\,N$ to substituting into M (which may be more complex then $x\,N$), we have reduced the type of the variable we are substituting for from $A_1 \to A_2$ to A_1.

This is analogous and was in fact inspired by an analogous structural measure in the proof of the admissibility of cut in the sequent calculus [Pfenning 2000], which proceeds by nested induction, first on the structure of the cut formula and second on the structure of derivations being cut.

In the formal development it is simpler if we can stick to the structure of the example above and use only non-dependent types in hereditary substitutions. We therefore first define type approximations α and an erasure operation $()^-$ that removes dependencies. Before applying any hereditary substitution $[M/x]^a_A(B)$ we first erase dependencies to obtain $\alpha = A^-$ and then carry out the hereditary substitution formally as $[M/x]^a_\alpha(B)$. A similar convention applies to the other forms of hereditary substitutions.

$$\begin{array}{rrcl} \text{Type approximation} & \alpha,\beta & ::= & a \mid \alpha \to \beta \mid \alpha[\gamma] \Rightarrow \beta \\ \text{Context approximation} & \gamma,\psi & ::= & \cdot \mid \gamma, x{:}\alpha \mid \gamma, x{:}\_ \end{array}$$

The last form of context approximation, $x{:}\_$ is needed when the approximate type of x is not available.[2] It does not arise directly from erasure.

Types and contexts relate to type and context approximations via an erasure operation $()^-$ which we overload to work on types and contexts.

$$\begin{array}{lcl} (a)^- & = & a \\ (\mathsf{app}(P,N))^- & = & P^- \\ (\mathsf{mapp}(P,\hat{\Psi}.N))^- & = & P^- \\ (\Pi x{:}A.\,B)^- & = & A^- \to B^- \\ (\Pi u{::}A[\Psi].\,B)^- & = & A^-[\Psi^-] \Rightarrow B^- \\ (\cdot)^- & = & \cdot \\ (\Psi, x{:}A)^- & = & \Psi^-, x{:}A^- \end{array}$$

We can now define $[M/x]^n_\alpha(N)$ and $[M/x]^r_\alpha(R)$ by nested induction, first on the structure of the type approximation α and second on the structure of the objects N and R. In other words, we either go to a smaller type approximation (in which case the objects can become larger), or the type approximation remains the same and the objects become smaller. We write $\alpha \le \beta$ and $\alpha < \beta$ if α occurs in β (as a

[2] See the definition of $[\sigma]^n_\psi(\mathsf{lam}(y.\,M))$ in the electronic appendix.

proper subexpression in the latter case). Such occurrences can be inside a context approximation ψ in the modal approximation $\beta_1[\psi] \Rightarrow \beta_2$, so we also write $\alpha < \psi$ if $\alpha \leq \beta$ for some $y{:}\beta$ in ψ, and we write $\alpha < \beta[\psi]$ if $\alpha \leq \beta$ or $\alpha < \psi$.

If the original term is not well-typed, a hereditary substitution, though terminating, cannot always return a meaningful term. We formalize this as failure to return a result. Later we show that on well-typed terms, hereditary substitution will always return well-typed terms.

As a convention when discussing expressions that may fail to be defined, whenever we state an equality $T_1 = T_2$, we imply that T_1 and T_2 are both defined and equal.

The LF Fragment. We start with the LF fragment to highlight the basic ideas in the simpler case before adding meta-variables. We have the following operations, where the superscript indicates the domain of the operation.[3]

$$\begin{array}{llll} [M/x]^n_\alpha(N) = N' & & & \text{Hereditary substitution into } N \\ [M/x]^r_\alpha(R) = R' & \text{or} & M' : \alpha' & \text{Hereditary substitution into } R \\ [M/x]^a_\alpha(A) = A' & & & \text{Hereditary substitution into } A \\ [M/x]^p_\alpha(P) = P' & & & \text{Hereditary substitution into } P \end{array}$$

Any of these functions can implicitly fail, which is propagated to the top level. Substitution $[M/x]^r_\alpha$ into an atomic term could return either an atomic term or a normal term, depending on the variable at the head of the term. If the variable at the head is different from x it returns an atomic term. If it is equal to x it must call substitution hereditarily, in which case the result is normal. In the latter case we also return a type approximation $\alpha' \leq \alpha$. One further remark: where hereditary substitution $[M/x]^*_\alpha$ is invoked in the typing judgment, we have $M \Leftarrow A$ for some A such that $\alpha = (A)^-$. As we recurse we maintain this invariant.

The cases for substitution into a normal term are just compositional.

$$\begin{array}{lcll} [M/x]^n_\alpha(\mathsf{lam}(y.\, N)) & = & \mathsf{lam}(y.\, N') & \text{where } N' = [M/x]^n_\alpha(N) \\ & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\ [M/x]^n_\alpha(R) & = & M' & \text{if } [M/x]^r_\alpha(R) = M' : \alpha' \\ [M/x]^n_\alpha(R) & = & R' & \text{if } [M/x]^r_\alpha(R) = R' \\ [M/x]^n_\alpha(N) & & \text{fails} & \text{otherwise} \end{array}$$

Because of compositionality, the hereditary substitution may fail to be defined only if the substitutions into the subterms are undefined. The side conditions $y \notin \mathsf{FV}(M)$ and $y \neq x$ do not cause failure, because they can always be satisfied by appropriately renaming y.

Substitution into an atomic term may fail to be defined depending on what is

[3]We use n for normal object, r for atomic object, a for normal type, p for atomic type, and s for substitution; k for kind is used earlier but its straightforward definition is omitted. We also use γ for contexts and δ for modal contexts. In general, we write $*$ in the substitution $[M/x]^*_\alpha$ to denote n, r, a, etc.

returned as a result of substituting into the function part of an application.

$$
\begin{array}{lcll}
[M/x]^r_\alpha(x) & = & M : \alpha & \\
[M/x]^r_\alpha(y) & = & y & \text{for } y \neq x \\
[M/x]^r_\alpha(\mathsf{app}(R, N)) & = & \mathsf{app}(R', N') & \text{if } [M/x]^r_\alpha(R) = R' \text{ and } N' = [M/x]^n_\alpha(N) \\
[M/x]^r_\alpha(\mathsf{app}(R, N)) & = & M_2 : \alpha_2 & \text{if } [M/x]^r_\alpha(R) = \mathsf{lam}(y.\, M') : \alpha_1 \to \alpha_2 \\
 & & & \text{where } \alpha_1 \to \alpha_2 \leq \alpha \text{ and } N' = [M/x]^n_\alpha(N) \\
 & & & \text{and } M_2 = [N'/y]^n_{\alpha_1}(M') \\
[M/x]^r_\alpha(\mathsf{app}(R, N)) & & \text{fails} & \text{otherwise}
\end{array}
$$

For example, if $[M/x]^r_\alpha(\mathsf{app}(R, N))$ does not return an atomic term or a lambda expression, or if the returned approximation $\alpha_1 \to \alpha_2$ is not a subexpression of α, then the substitution into $\mathsf{app}(R, N)$ is undefined. Shortly, we will prove that $\alpha_1 \to \alpha_2$ will always be a subexpression of α so that the check is superfluous. We nevertheless include the check here to make it obvious that the hereditary substitution is well-founded, because recursive appeals to substitutions take place on smaller terms with equal approximation, or on a smaller approximation.

Since variables only range over objects, hereditary substitution for types A and atomic types P is trivially compositional. We only show these cases here and omit them in the rest of the development.

$$
\begin{array}{lcll}
[M/x]^a_\alpha(\Pi y{:}B.C) & = & \Pi y{:}B'.C' & \text{where } B' = [M/x]^a_\alpha(B) \text{ and } C' = [M/x]^a_\alpha(C) \\
 & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
[M/x]^a_\alpha(P) & = & P' & \text{where } P' = [M/x]^p_\alpha(P) \\
[M/x]^p_\alpha(a) & = & a & \\
[M/x]^p_\alpha(\mathsf{app}(P, N)) & = & \mathsf{app}(P', N') & \text{where } P' = [M/x]^p_\alpha(P) \text{ and } N' = [M/x]^n_\alpha(N)
\end{array}
$$

An even more straightforward definition applies for the substitution into kinds.

The Full Language. Following the above idea and the definition of non-hereditary substitution, it is not difficult to extend the definitions to the full language. We present the complete definition of hereditary substitution in the electronic appendix, using the notation

$$
\begin{array}{l}
[M/x]^*_\alpha(\_) \\
[\![\hat{\Psi}.M/u]\!]^*_{\alpha[\psi]}(\_) \\
[\sigma]^*_\psi(\_)
\end{array}
$$

for $* = n, r, s, a, p, k, \gamma, \delta$ and $\_ = N, R, \sigma, A, P, K, \Gamma, \Delta$, respectively.

To illustrate the extension, we present here a characteristic fragment of the definition of hereditary modal substitution, that is, the substitution for a meta-variable. We will concentrate on the case where we apply a hereditary modal substitution to an atomic term $[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R)$. The result of this substitution is either an atomic term R' or a normal term $M' : \alpha'$ with a type approximation.

In the definition of this substitution, we require that $\hat{\Psi}$ lists exactly the variables from the context approximation ψ which appears as a bounding index of the substitution. Because $\hat{\Psi}$ in $\hat{\Psi}.M$ is a binder and its variables can be renamed, this requirement really expresses that the number of bound variables in $\hat{\Psi}.M$ matches the number of variables in the context approximation ψ.

We highlight the most interesting cases, namely when substituting into a meta-variable closure $\mathsf{clo}(u, \sigma)$ and a modal application $\mathsf{mapp}(R, \hat{\Gamma}.N)$.

First, the cases where we have reached a closure. If the meta-variable at the head is the same as we are substituting for, we need to invoke a hereditary simultaneous substitution, similar to the non-hereditary meta-variable substitution in Section 4.3.

$$\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(\mathsf{clo}(u,\tau)) & = & M' : \alpha & \text{where } \tau' = [\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
 & & & \text{and } M' = [\tau'/\hat{\Psi}]^n_{\psi}(M) \\
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\Psi]}(\mathsf{clo}(v,\tau)) & = & \mathsf{clo}(v,\tau') & \text{if } v \neq u \\
 & & & \text{with } \tau' = [\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\Psi]}(\mathsf{clo}(v,\tau)) & & \text{fails} & \text{otherwise}
\end{array}$$

The failure case only applies when one of the appropriate recursive calls fails.

For an application, the result differs depending on whether the head of R is equal to the meta-variable we are substituting for or not. If so, then substitution needs to proceed hereditarily and we return a normal term together with it approximate type. If not, we just substitute recursively and re-build an application.

$$\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(\mathsf{mapp}(R,\hat{\Gamma}.N)) & = & M_2 : \alpha_2 & \text{if } [\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R) = \\
 & & & \quad \mathsf{mlam}(v.\, M') : \alpha_1[\gamma] \Rightarrow \alpha_2 \\
 & & & \text{where } \alpha_1[\gamma] \Rightarrow \alpha_2 \leq \alpha[\psi] \\
 & & & \text{and } \hat{\gamma} = \hat{\Gamma} \\
 & & & \text{and } N' = [\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) \\
 & & & \text{and } M_2 = [\![\hat{\Gamma}.N'/v]\!]^n_{\alpha_1[\gamma]}(M') \\
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(\mathsf{mapp}(R,\hat{\Gamma}.N)) & = & \mathsf{mapp}(R',\hat{\Gamma}.N') & \text{if } [\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R) = R' \\
 & & & \text{and } N' = [\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(\mathsf{mapp}(R,\hat{\Gamma}.N)) & & \text{fails} & \text{otherwise}
\end{array}$$

In the above cases we do not substitute into $\hat{\Gamma}$, as $\hat{\Gamma}$ is just a list of ordinary variables without any type information. In the first case, we need to verify that $\hat{\gamma} = \hat{\Gamma}$ in order to establish our invariant to allow the final computation of M_2. Since $\hat{\Gamma}.N$ binds the variable in $\hat{\Gamma}$ and therefore allows their renaming, this requires only that γ and $\hat{\Gamma}$ have the same length. If not, hereditary substitution will fail, as it will in the case where the result of substituting into R returns a normal term which is not a meta-variable abstraction.

5.2 Properties of Hereditary Substitutions

The first property states that the hereditary substitution operations terminate, independently of whether the terms involved are well-typed or not.

THEOREM 5.1 TERMINATION.

(1) *If* $[M/x]^r_{\alpha}(R) = M' : \alpha'$ *then* $\alpha' \leq \alpha$

(2) $[M/x]^*_{\alpha}(\_)$, $[\![\hat{\Psi}.M/u]\!]^*_{\alpha[\psi]}(\_)$ *and* $[\sigma]^*_{\psi}(\_)$ *terminate, either by returning a result or failing after a finite number of steps.*

PROOF. The first part is verified by induction on the definition of all operations. The second part follows by a nested induction, first on the structure of α, $\alpha[\psi]$

and ψ and second on the structure of the term we apply hereditary substitution to. Note that type approximations contain context approximations and vice versa, so the structural ordering on these is meaningful and well-founded. □

Termination is the key property allows us to disentangle the mutual dependency between typing and normalization which is the most difficult obstacle to overcome in the meta-theory of dependent type theories.

THEOREM 5.2 DECIDABILITY OF TYPE CHECKING.
All judgments in the dependent contextual modal type theory are decidable.

PROOF. The typing judgments are syntax-directed and therefore clearly decidable. Hereditary substitution always terminates, giving us a decision procedure for dependent typing. □

It is remarkable that we can prove decidability even without proving that hereditary substitution of well-typed terms into well-typed terms again yields well-typed terms. Nonetheless, such a theorem is critical for the logical framework methodology, and to relate our formulation of the type theories to others in the literature. We state here only the version of the theorem needed to validate the operations that arise during type-checking; more details and appropriate generalizations can be found in the electronic appendix.

THEOREM 5.3 HEREDITARY SUBSTITUTION PRINCIPLES FOR TYPES.
Assume the contexts in the given judgments below are well-formed.

(*1*) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A \vdash B \Leftarrow \mathsf{type}$ *then* $\Delta;\Gamma \vdash [M/x]^a_{A^-}(B) \Leftarrow \mathsf{type}$.

(*2*) *If* $\Delta;\Psi \vdash M \Leftarrow A$ *and* $\Delta, u{::}A[\Psi];\Gamma \vdash B \Leftarrow \mathsf{type}$
then $\Delta;[\![\hat{\Psi}.M/u]\!]^a_{A^-[\Psi^-]}(\Gamma) \vdash [\![\hat{\Psi}.M/u]\!]^a_{A^-[\Psi^-]}(B) \Leftarrow \mathsf{type}$

(*3*) *If* $\Delta;\Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta;\Psi \vdash B \Leftarrow \mathsf{type}$ *then* $\Delta;\Gamma \vdash [\sigma]^a_{\Psi^-}(B) \Leftarrow \mathsf{type}$.

PROOF. Direct consequence of the hereditary substitution principles given in the electronic appendix. In brief, we generalize the judgments over all syntactic categories and then proceed by nested induction on the index of the hereditary substitution (A^-, $A^-[\Psi^-]$, and Ψ^-, respectively) and the structure of the derivation we substitute into. □

For more detail on the technique of hereditary substitutions and discussion of related issues we refer the reader to the technical report by Watkins et al. [2002]. Simultaneous and modal substitution add some bulk to the development, but do not require any essentially new ideas not already introduced above.

6. STAGED FUNCTIONAL COMPUTATION

In this section we show how contextual modal type theory can avoid generating spurious redexes in staged computation and run-time code generation, producing simpler and more efficient code than is possible with validity alone.

Davies and Pfenning [2001] have proposed the use of the modal necessity operator $\Box$ in order to provide type-theoretic support for staged computation and, more specifically, run-time code generation. Abstractly, values of type $\Box A$ have the form

$\mathsf{box}(M)$, where M is an (unevaluated) source expression. In other words, the box constructor functions like a quotation operator. The introduction rule

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathsf{box}(M) : \Box A}\ \Box\mathsf{I}$$

enforces that M does not contain any free ordinary variables x, only other meta-variables in Δ. In the implementation of this calculus for run-time code generation [Wickline et al. 1998], a value $\mathsf{box}(M)$ of type $\Box A$ is actually not maintained as a source expression, but compiled to a generator which will produce compiled code for M at run-time. The modal restriction means that programs are correctly staged, that is, all the information to generate efficient code from M is indeed available when the generator is invoked. Abstractly, the elimination rule

$$\frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, u{::}A; \Gamma \vdash N : C}{\Delta; \Gamma \vdash \mathsf{let\,box}\, u = M\ \mathsf{in}\ N\ \mathsf{end} : C}\ \Box\mathsf{E}$$

binds u to the source expression that M evaluates to and substitutes it in N. Variables $u{::}A$ then refer to source expressions.

$$\frac{u{::}A\ \mathsf{in}\ \Delta}{\Delta; \Gamma \vdash u : A}\ \mathsf{mvar}$$

In the implementation, an occurrence of u in N inside a box operator corresponds to the inner generator invoking the outer one during its operation, while an occurrence of u in N not underneath a box will invoke the generator and then jump to it, executing the generated code.

As mentioned earlier, in the syntax of this paper, $\Box A$ would be written as $[\cdot]A$. Proof terms are also simple notational variants: $\mathsf{box}(M)$ would be $\mathsf{box}((\cdot).\, M)$ and $\mathsf{let\,box}\, u = M\ \mathsf{in}\ N\ \mathsf{end}$ would be $\mathsf{letbox}(M, u.\, N)$, and variables u become $\mathsf{clo}(u, \cdot)$.

As a simple standard example, consider the exponentiation function $\mathsf{exp}\ n\ b$, which computes b^n. We present the examples in a syntax reminiscent of Standard ML, including integers, some integer operations, and recursion. We also do not provide a formal operational semantics, but it is a straightforward call-by-value functional language based on the reductions $\Longrightarrow_R$ from Section 4.4. In this setting box is akin to a quotation operator and we do not evaluate in its scope.

```
(* val exp : int -> (int -> int) *)
fun exp 0 = fn b => 1
  | exp n = fn b => b * exp (n-1) b
```

This function will always return a functional object (that is, a closure in the implementation) after one step.

We can stage the computation in such a way that when given the exponent n it generates a function to compute b^n. Using modal types we can not only enforce this staging as shown below, we can also implement it so that it returns a source expression (or its corresponding generator) rather than just a closure.

```
(* val exp : int -> [](int -> int) *)
fun exp 0 = box (fn b => 1)
  | exp n = let box u = exp (n-1) in box (fn b => b * u b) end
```

We then have (using some variable renaming for readability)

```
exp 2 ==> box (fn b2 => b2 * (fn b1 => b1 * (fn b0 => 1) b1) b2)
```

If we carry out some variable-for-variable reductions, we can see that the output represents a somewhat less direct version of

```
box (fn b2 => b2 * (b2 * 1))
```

which would avoid unnecessary function calls and is therefore the ultimately desired residual expression.

The logical view of staged computation proposed above is simple, elegant, and effective [Wickline et al. 1998]. When used as a mechanism for meta-programming, however, it has the frequently cited drawback [Davies 1996; Taha and Sheard 1997] that the expressions that are manipulated must be closed. The example of `exp 2` shown above is one example where the residual code could be improved further if one could manipulate expressions with free variables as in work by Davies [1996] and recent research on MetaML [Calcagno et al. 2004] which are based on temporal logic instead of the modal logic of validity. Moving to temporal logic, however, comes at a price, because we still need to track closed expressions if we want to be able to evaluate them explicitly, which accounts for a good deal of the complexity of the MetaML type system.

Instead of using temporal type theory we can overcome these difficulties by generalizing from modal logic to contextual modal logic. Recall from Sections 2.4 and 4.1 that contextual modal logic is an extension of the modal logic proposed above where $\Box A$ corresponds exactly to $[\cdot]A$. We can now reformulate the function of type `int -> [](int -> int)` into a function of type `int -> [b:int]int`, where the output is closed *except* that it can mention one variable (named `b` in the type).

In the concrete syntax, the `box` operator now is itself a binding operator (instead of taking a function as an argument), where we write `box b1,...,bn => M` for $\mathsf{box}(b_1, \ldots, b_n.\, M)$. We write a substitution as a tuple `[M1,...,Mn]` which is applied to a meta-variable u by position, rather then by name in the form `u[M1,...,Mn]`. Below is the improved exponentiation function.

```
(* val exp : int -> [b:int]int) *)
fun exp 0 = box b => 1
  | exp n = let box u = exp (n-1) in box b => b * u[b] end
```

When substituting an expression M for u in $u\ b$ in the previous version, we generate a residual function application $M\ b$. Here we substitute $[\![(b_2.M)/u]\!](u\ [b_1/b_2])$ which directly applies the substitution $[b_1/b_2]$ to M. In this manner, we directly obtain the desired residual expression.

```
exp 2 ==> box (b => b * (b * 1))
```

Other approaches, such as Taha and Nielsen's environment classifiers [Taha and Nielsen 2003], may be more parsimonious on some examples and perhaps more amenable to type inference (which we do not consider here), but they come at a high price in syntactic and conceptual complexity. One of the consequences of this complexity is that it is difficult to see how one might formally reason about the correctness of staged programs. Our type-theoretic setting could provide an

advantage in that respect since it is consistent with dependent types. A brief discussion of this possibility can be found among the future work in Section 9.2.

We close this section with the observation that we can convert between the two forms of abstraction.

```
(* val f : [](A -> B) -> [y:A]B *)
fun f x = let box u = x in box y => u[] y end

(* val g : [y:A]B -> [](A -> B) *)
fun g z = let box v = z in box (fn y => v[y]) end
```

The function to evaluate an expression still requires its argument to be closed, when the language is pure.

```
(* val eval : []A -> A *)
fun eval x = let box u = x in u[] end
```

However, when the language has non-termination, exceptions, or other expression of arbitrary type, we can write, for example,

```
(* val eval' : [y:B]A -> A *)
fun eval' x = let box u = x in u[raise exn] end
```

Then `eval' (box y => M)` raises the exception `exn` if the evaluation of M would refer to y.

7. IMPLEMENTATION OF META-VARIABLES IN LOGICAL FRAMEWORKS

In this section, we sketch how dependent contextual modal type theory as presented in Section 5 can be employed for the efficient implementation of meta-variables in a logical framework. This is a different form of application than given in the previous section, not only because we rely on canonical forms, but also because explicit quantification over meta-variables is at present not available to the user, only in the implementation. More details on this application in the context of the Twelf system [Pfenning and Schürmann 1999] are given by Pientka [2003] and Pientka and Pfenning [2003]. For the non-dependent case, similar observations (without any logical foundations) have been made by Dowek et al. [1995; 1996] and later generalized by Muñoz [2001].

An implementation of a logical framework must handle meta-variables, which are used both during type reconstruction and during proof search. These meta-variables are placeholders for terms in the language under consideration, to be determined or constrained by unification, proof search, or perhaps even directly by the user. Since dependent type theories reify proofs as objects, unsolved subgoals can also be represented as meta-variables.

When the value of a meta-variable is determined we say that the meta-variable is *instantiated* in order to distinguish this from the process of substitution of a term for an ordinary variable. To avoid misunderstandings, we will refer to ordinary variables as *parameters*. In contrast to meta-variables, these should be interpreted universally, that is, they are not subject to instantiation.

There are four principal questions in the handling of meta-variables:

(1) Which parameters (that is, bound or universally quantified variables) are allowed to occur in a term that instantiates a meta-variable?
(2) What constraints does the type theory impose on occurrences of meta-variables in the contexts or types of other meta-variables?
(3) How do we implement meta-variables and the instantiation operation?
(4) Which algorithm do we use for unification or constraint simplification?

The first three questions apply, regardless of whether the meta-variables stand for open subgoals during proof search, or if they are placeholders for terms to be determined by unification.

The last one is beyond the scope of this paper, although Dowek et al. [1995; 1996] and Pientka [2003] provide answers that are consistent with our approach. For the first three questions, dependent contextual modal type theory provides clean and simple answers that have been experimentally validated in Twelf.

Parameter Occurrences. The question which parameters may occur in the term that instantiates a meta-variable is answered quite directly by our type theory. A meta-variable $u{::}A[\Psi]$ can depend exactly on the parameters in Ψ. All occurrences of u in a term have the form $\mathsf{clo}(u,\sigma)$ with the typing rule

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \mathsf{clo}(u,\sigma) \Rightarrow [\sigma]^a_\Psi(A)}\ \mathsf{mvar}$$

This means that σ is a mediating substitution, providing a translation from the context Ψ to the ambient context Γ in effect where u occurs. Different occurrences of u may be under different mediating substitutions σ, avoiding possible confusions of variable names. While this means of achieving consistency may appear to be somewhat heavy, it is in fact quite efficiently implementable by using explicit substitutions as sketched below.

Meta-Variable Occurrences. Recall the rules to verify that modal contexts are well-formed.

$$\frac{}{\vdash \cdot\ \mathsf{mctx}} \qquad \frac{\vdash \Delta\ \mathsf{mctx} \quad \Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta; \Psi \vdash A \Leftarrow \mathsf{type}}{\vdash (\Delta, u{::}A[\Psi])\ \mathsf{mctx}}$$

This means that we must be able to present all meta-variables that appear in a given term in a linear order so that the types and contexts of meta-variables further to the right may mention meta-variables further to the left. Since the set of currently available meta-variables is usually not made explicit in an implementation, we had missed this invariant in an earlier implementation of unification for the LF logical framework. Fortunately, in the case of higher-order patterns in the sense of Miller [1991], this restriction does not place any additional burden on the implementation since the steps of the algorithm implicitly maintain this invariant [Pientka 2003]. However, when constraints outside the pattern fragment are generated and maintained, additional work during an occurs-check may be necessary to enforce this invariant.

There is a related question: which occurrences of *meta-variables* in the substitution term for another meta-variable are sound. In the first-order, non-dependent

case this reduces to a simple occurs-check. In this situation, the question is most easily answered by analyzing simultaneous substitution for meta-variables. This can easily be done in the framework set out here. The details are beyond the scope of this paper and can be found in Pientka's Ph.D. thesis [2003].

Implementation Considerations. The type theory suggests certain implementation choices which we briefly discuss here. The first is dictated by the fact that closures $\mathsf{clo}(u, \sigma)$ are part of the syntax of canonical forms. Consequently, the implementation should support substitutions as explicit syntactic objects. In order to avoid the excessive renaming inherent in the substitutions $[\![\hat{\Psi}.M/u]\!]^*_{A[\Psi]}$ we are led to a representation using de Bruijn indices, since multiple renamings of a context Ψ are identical. Moreover, the type-free context $\hat{\Psi}$ in $[\![\hat{\Psi}.M/u]\!]$ becomes unnecessary, since it only records a bound on the de Bruijn index of variables free in M. But these can be obtained from the type $u{::}A[\Psi]$, so they do not need to be maintained during substitution.

The next critical observation is that in the absence of abstraction over meta-variables, the substitution operation (hereditary or otherwise) for a meta-variable does not need to check *any* condition related to bound variables (please see the electronic appendix). This would suggest to represent meta-variables directly by pointers, and to instantiate meta-variables by updating these pointers, called *grafting* by Dowek et al. [1995; 1996]. Clearly, since each variable occurs as part of a closure $\mathsf{clo}(u, \sigma)$ with an appropriate mediating substitution σ, updating u in place does not destroy any typing or variable naming invariant.

However, there is a catch: our system only treats canonical forms (the only terms of interest in a logical framework), the result of replacing u by a term M would no longer be canonical and not even be syntactically meaningful. The solution is to adopt a notion of *postponed hereditary substitution* directly modeled after the notion of *explicit substitution* [Abadi et al. 1990]. However, instead of giving postponed hereditary substitutions first-class status and investigate properties such as strong normalization or the Church-Rosser property, we view them only as an implementation technique. That means we should never directly analyze terms such as $\mathsf{clo}(M, \sigma)$ unless M is a meta-variable, and all operations should propagate the substitution inwards. One can think of this as a lazy implementation of the hereditary substitution operations detailed in Section 5. With this change, grafting via pointer-update is a correct implementation of instantiation as given by $[\![\hat{\Psi}.M/u]\!]$.

This notion of lazy hereditary substitution provides a clean and simple organizing principle for many operations on terms in the Twelf implementation. These operations come in two forms: one operating on a weak head-normal form, and one operating on a term with a possible postponed hereditary substitution which is pushed downwards far enough to expose a term in weak head-normal form. As we traverse the term, we therefore expose the canonical term layer by layer, carrying out only as much of the hereditary substitution as necessary. A similar technique has proved quite effective in λProlog, although both the formal basis and the implementation differs in some important aspects [Liang and Nadathur 2002]. A further discussion is beyond the scope of this paper, but some additional material can be found in a related paper [Pientka and Pfennning 2003].

8. RELATED WORK

We roughly divide the related work into research on modal logic, functional programming, and meta-variables in theorem proving.

8.1 Modal logics of context

The formal notion of context has been extensively researched for various applications in artificial intelligence. One of the early proposals is by Weyhrauch [1979], motivated by Weyhrauch's work in interactive theorem proving. Later, McCarthy [1987; 1993], proposed a logic of context as a solution to the problem of generality in common-sense reasoning. McCarthy's premise is that logical axioms are usually not general, but crucially depend on the context in which they are asserted. Thus, a formalization of the notion of context is a necessary first step in formulating general axioms for common-sense knowledge, because it facilitates a precise expression of the different meanings that a logical statement can take in different contexts. This line of research eventually led to the formulation of the *propositional logic of contexts* (PLC) by Buvač et al. [1995], and an extension to first-order logic [Buvač 1996]. In these theories, contexts are first class objects (i.e., the theory provides terms that denote contexts), and are modeled by sets of propositions. Each context can make statements about the truth of formulas in other contexts, by using the modal operator *ist*. For example, in a context c_1 we can assert the formula $ist(c_2, A)$, establishing that, when viewed from c_1, the proposition A *is true* in the context c_2.

The work of Giunchiglia on contextual reasoning [Giunchiglia 1993] is motivated by an observation that reasoning about a certain problem usually involves only a small subset of the overall available knowledge. Thus, Giunchiglia proposes that the knowledge be organized into contexts – modular, but related units, that formulate *local theories*, which any particular reasoning agent may or may not employ. As a realization of this idea, we mention the work on *multi-context systems* [Giunchiglia and Serafini 1994; Serafini and Giunchiglia 2002]. In multi-context systems, contexts represent the propositions associated with the knowledge or belief of a particular agent. Each context is therefore labeled by a unique identifier of the agent it represents. The multi-context systems are multi-lingual, meaning that facts from different contexts can be represented using different languages, and there is an apparatus for specifying translations between the languages. Facts that span multiple contexts are specified via so-called *bridge rules*, which are inference rules whose premises and conclusions belong to different languages. Technically, multi-context systems are founded in the multi-modal version of the modal logic K. For a formal relationship between PLC and multi-context systems, we refer to a paper by Serafini and Bouquet [2004].

Attardi and Simi [1995; 1994] propose the notion of a viewpoint as means of expressing relativized truth. A viewpoint is a set of propositions representing assumptions of a theory of interest. For example, the sentence $\mathrm{in}('A', vp)$ states that the proposition A is true in the context of a viewpoint vp. The operator *in* for entailment in a viewpoint does not modify the proposition A, but rather, expects the syntactic representation of A (here denoted as $'A'$). The syntactic nature of entailment in a viewpoint is exploited in the representation of other modal operators

like knowledge and belief, which are viewed as first-order predicates over *syntactic sentences*, rather than as operators on propositions. The process of entering a new viewpoint is called reification, and exporting a knowledge from one viewpoint into another is called reflection. A proposition can be reified only if it is derived in a restricted way, without using reflection. This restriction is motivated by the inconsistency that arises in formal systems capable of both representing their own syntax, and also reasoning by reflection [Montague 1963].

A type theoretic approach to the formulation of the logic of contexts has been advocated by Thomason [1999; 2003]. In addition to the types of individuals, truth-values, and functions, this type theory features the type w of *possible worlds*. There are no constants or variables of type w, so there is no explicit lambda abstraction over worlds. Instead, the type theory contains operators for forming *intensions*, which are functions over possible worlds. For example, if $e{:}A$ is an expression, then $\hat{e}$ is an intensional expression and has type $w \to A$. Dually, if e is an intensional expression of type $w \to A$, then $\check{e}$ is its *extension*, and has type A. The type theory further contains the type of *indices*, whose elements denote differently in different contexts. Typical examples are the expression "I" and "here", which, obviously, have different meanings depending on the context of the person that interprets them. Indices can be built into more complex types, like the type of context-dependent propositions, or the type of context-dependent intensions.

Finally, we mention the work of de Paiva [2003], who studies the relationship between several known propositional logics of contexts, like the PLC logic and multi-context systems mentioned above, and proposes a constructive version of the multi-modal K as the common foundation. The paper further discusses the proof term assignment for this logic, and the corresponding Curry-Howard isomorphism, along the lines of Alechina et al. [2001]. This proof term assignments is different from ours in that it does not separate the ordinary from the meta-variables. As a consequence, it has to employ explicit substitutions even when the modalities are not indexed by contexts.

8.2 Context in functional programming

In functional programming, various formulations of context as a primitive programming construct have been considered [Sato et al. 2001; Sato et al. 2002; Mason 1999; Hashimoto and Ohori 2001], with applications in explaining binding structure and dynamic binding [Dami 1998], and in incremental program construction [Lee and Friedman 1996].

For example, the $\lambda\kappa\epsilon$-calculus of Sato et al. [2002], allows a simultaneous abstraction over a set of variables. The expression $\kappa\{\Psi\}.\, M$ abstracts the variables listed in Ψ from the expression $M : A$. The type of $\kappa\{\Psi\}.\, M$ is A^{Ψ}, similar to our type $[\Psi]A$. There are many distinctions, however, between $\lambda\kappa\epsilon$ and the proof terms for ICML, arising mostly because $\lambda\kappa\epsilon$ is not based on modal logic. For example, the context in $\lambda\kappa\epsilon$ associates variables with types, but not with the context that they depend on. This leads to a somewhat complicated formulation, where each variable must be assigned an integer level, and the typing rules and the operation of substitution must perform arithmetic over levels. Another difference is that $\lambda\kappa\epsilon$-calculus admits outside free variables to appear in the body of a context abstraction. This is prevented in ICML by the typing rules for $[\Psi]A$, and is essential

for the modeling and implementation of meta-variables. As explained in Section 7, substitutions for meta-variables do not need to check any side conditions regarding bound variables, allowing meta-variables to be represented by pointers. Without the modal restriction, this aspect of meta-variables would not have been captured.

The $\lambda\epsilon$-calculus of Sato et al. [2001] is a precursor to $\lambda\kappa\epsilon$. The $\lambda\epsilon$-calculus provides explicit substitution of terms for variables, but a variable may be used only if it is bound by the explicit substitution.

Mason [1999] extends the untyped λ-calculus with a primitive notion of context, and the related operations for declaring and filling context holes. Holes are similar to our meta-variables, in the sense that each hole is decorated with its corresponding substitution. Mason considers operations of strong and weak application of variable substitutions. Both operations propagate the substitution down to the holes, but differ in their action at the holes. Strong application composes the substitution with the hole's substitutions (possibly changing the domain of the hole's substitution), while weak application only distributes over the terms of the hole's substitution. Mason's operation of hole filling correspond to our modal substitution, in the sense that the substitution decorating each hole is applied (weakly or strongly) over the term being placed into the hole. The paper does not consider abstraction over holes, nor is there a term constructor similar to our box that introduces a modal distinction between terms.

Hashimoto and Ohori [2001] present a typed calculus which internalizes the notion of computation in context via a type of functions from contexts to values. The calculus distinguishes between ordinary variables and hole variables. Hole variables correspond to our meta-variables; they are typed in a separate hole context Δ which associates with every hole variable u a type A, an interface Ψ (roughly corresponding to our context), but also an explicit substitution σ that specifies the bindings of the hole. Explicit substitutions only rename variables with other variables. Associating holes and substitutions in Δ complicates the system significantly and reduces its expressiveness. For example, each hole variable can only be associated with one substitution, and the typing rules must non-trivially manipulate the hole context. Even more severely, the hole context must be linear, that is, hole variables can only be used once, and ordinary variables can be referenced only when the hole context is empty.

The mentioned works are computationally motivated, and typically do not explore the logical aspects of contexts. A modal logic of contexts as a foundation for functional programming with various kinds of effects has recently been proposed by Nanevski [2003; 2004]. In that work, modal validity classifies computations that are effectful in the sense that the computation results depend on the run-time environment, but execution does not change the run-time environment itself. Examples include programs that read (but do not write) from memory, or programs with control-flow effects. Computations that may change the run-time environment (for example, a computation that writes into memory) are ascribed a type corresponding to modal possibility. Modal types for effectful computations are indexed by sets of *names*, where each name corresponds to a particular effect instance. In this respect, sets of names are similar to context from the current paper. There are significant differences between the two, however; the principal difference being

that names have global identity. One and the same name can appear in the index of many different modal operators. In contrast, a variable bound in a context has local scope and is thus necessarily different from any other bound variable in the program.

A modal λ-calculus has been proposed as a foundation for staged computation and metaprogramming by Davies and Pfenning [2001], and was subsequently used as a language for run-time code generation by Wickline et al. [1998; 1998]. In this work, the modal constructor $\Box A$ is used to classify *closed* code of type A. Closed code contains no free ordinary variables, and is thus executable at run time. The applicability of closed code is sometimes restrictive, so alternative approaches for typing *open* code were developed; most significantly, $\lambda^{\bigcirc}$ [Davies 1996], and MetaML [Taha and Sheard 1997]. Both of these calculi are based on a proof term system for a variant of temporal logic with a modality expressing truth at the next time moment. From the computational perspective, the modal type classifies open code, but the code expressions cannot be evaluated at run time unless further guarantees are provided. For some time, the focus of the work on staged computation and metaprogramming has been in combining these two approaches. Taha and Nielsen [2003] and Calcagno et al. [2004] achieve this by relying on environment classifiers, which stand for an unnamed set of variables that a term may be open in. These calculi contain the types of code open in a set of environment classifiers, but also the operations of universal quantification over classifiers. Another approach to typing open code is the $\nu^{\Box}$-calculus developed by Nanevski and Pfenning [2005], where names stand for free variables, and there is also universal quantification over sets of names. The $\nu^{\Box}$-calculus is similar to the calculus presented here, except that in $\nu^{\Box}$ modalities are indexed by sets of names rather than by contexts.

8.3 Context in theorem proving

In general we can find two distinct motivations for context calculi in theorem proving. As discussed in the previous section, context calculi provide a foundation for meta-variables which need to be instantiated during proof search via higher-order unification. The concise description of higher-order unification requires a clear distinction between bound variables and meta-variables and efficient implementation techniques rely on instantiation of meta-variables which allows capture. One of the first approaches to allow first-order replacement and avoid the aforementioned problems in the simply-typed setting was developed by Dowek et al. [1995; 1996]. It relies on de Bruijn indices and explicit substitutions and was later extended to dependently typed theories by Muñoz [2001; 2000]. In this approach explicit substitutions can be associated with any term, not only with meta-variables. First-order replacement is achieved in two steps. First, terms are pre-cooked in such a way that scoping constraints are preserved. The main idea is, if some meta-variable X occurs under some λ-abstractions, then substituting M for X needs some processing called lifting to avoid capture of bound variables in M. Second, we can use first-order instantiation, called grafting. Although the use of de Bruijn indices leads to a simple formal system and it is very useful in an implementation, nameless representation of variables via de Bruijn indices are usually hard to read and critical principles are obfuscated by the technical notation.

Independently, Strecker [1999] developed a calculus with meta-variables as first-

class objects for a variant of the Extended Calculus of Construction. Similar to our approach, meta-variables are associated with a substitution to eliminate the need for raising and β-reduction during instantiation of meta-variables. However there are several differences. First of all, there is no explicit context for the meta-variables. As a consequence, it is difficult and it may be expensive to verify that an instantiation is valid, since meta-variable dependencies need to be computed afresh. Second, the substitutions attached to meta-variables are not simultaneous substitutions. As a result, the decidability of the typing rules is not directly obvious. Finally, the calculus lacks a logical foundation.

Context calculi not only play an important role in concisely describing meta-variables during higher-order unification, they also are an integral part of a foundation for interactive theorem proving where we need to reason about incomplete proofs or proof with holes [Magnusson 1995; Geuvers and Jojgov 2002; Jojgov 2003; Bognar and de Vrijer 2001]. The status of holes in proofs which may be filled later is essentially the same as meta-variables. Therefore, not surprisingly, many approaches build directly or indirectly on the treatment of meta-variables in higher-order unification and suffer the same problems and difficulties. For example, in Geuvers and Jojgov [2002], open terms are represented via a kind of meta-level Skolem function. However, in general reduction and instantiation of meta-variables (or holes) do not commute. This problem also arises in Bognar and de Vrijer [2001]. Our work resolves many of these aforementioned problems, since reduction and instantiation naturally commute and require no special treatment. Hence this work may also be viewed as a foundation for treating incomplete proofs.

Recently, Sato et al. [2003] proposed a calculus of meta-variables. Their approach is fundamentally different from the one presented in this paper. Each variable is given a level, which classifies the variables into bound variables (level 0), meta-variables (level 1) and meta-meta-variables (level 2) etc. Then the authors define a "textual" substitution which allows capture. There are two main obstacles with using this approach. First, since textual substitutions are not capture-avoiding, we will lose confluence. The second problem is that some reductions may get stuck. To address these problems the authors suggest to define reductions in such a way that it takes into account the different levels and keep track of arities of functions. This leads to a carefully engineered system which is confluent and strongly normalizing, although not very intuitive. Moreover, a formulation of algorithms such as higher-order unification with this calculus seems cumbersome.

9. EXTENSIONS AND FUTURE WORK

We briefly sketch three main avenues of future work: adding contextual possibility, defining a dependent necessity for reasoning about staged computation, and allowing explicit quantification over substitutions.

9.1 Contextual possibility

The contextual modal type theory can be extended to include a relativized version of modal possibility, and we have already presented some preliminary steps in that direction [Nanevski et al. 2003]. The idea is to introduce a new judgment $A\,\mathsf{poss}\langle\Psi\rangle$ which expresses existential quantification over possible worlds. The judgment $A\,\mathsf{poss}\langle\Psi\rangle$ holds if there *exists* a world in which Ψ and A are simultaneously

true. This is dual to relativized validity $A\,\mathsf{valid}[\Psi]$ which holds if A is true in *every* world in which Ψ is true. The judgment $A\,\mathsf{poss}\langle\Psi\rangle$ is internalized by means of a type operator $\langle\Psi\rangle A$, which degenerates into the customary operator for possibility $\Diamond A$ when Ψ is the empty context, The typing rules for relativized possibility are an indexed variant of the rules presented by Pfenning and Davies [2001]. We show here only the simply typed variant of the rules.

$$\frac{\Delta;\Gamma \vdash \sigma : \Psi \quad \Delta;\Gamma \vdash M : A\ \mathsf{true}}{\Delta;\Gamma \vdash \langle\sigma, M\rangle : A\ \mathsf{poss}\langle\Psi\rangle}\ \mathsf{poss}$$

$$\frac{\Delta;\Gamma \vdash E : A\ \mathsf{poss}\langle\Psi\rangle}{\Delta;\Gamma \vdash \mathsf{dia}(E) : \langle\Psi\rangle A\ \mathsf{true}}\ \Diamond\mathsf{I}$$

$$\frac{\Delta;\Gamma \vdash M : \langle\Psi\rangle A\ \mathsf{true} \quad \Delta;\Psi, x{:}A\,\mathsf{true} \vdash E : B\ \mathsf{poss}\langle\Theta\rangle}{\Delta;\Gamma \vdash \mathsf{letdia}(M, \langle\Psi, x\rangle.\, E) : B\ \mathsf{poss}\langle\Theta\rangle}\ \Diamond\mathsf{E}$$

$$\frac{\Delta;\Gamma \vdash M : [\Psi]A\ \mathsf{true} \quad \Delta, u{::}A\,\mathsf{valid}[\Psi];\Gamma \vdash E : B\ \mathsf{poss}\langle\Theta\rangle}{\Delta;\Gamma \vdash \mathsf{letbox}(M, u.\, E) : B\ \mathsf{poss}\langle\Theta\rangle}\ \Box\mathsf{E}^p$$

The additional rule $\Box\mathsf{E}^p$ is not related to possibility per se, but eliminates the validity modality into the possibility judgment. It is needed so that the system satisfies the subformula property [Pfenning and Davies 2001]. The rest of ICML can easily be extended to the new judgment. For example, contextual substitution simply commutes with all the new constructors. Extending simultaneous substitutions to the new cases is slightly more involved, as presented below.

$$\begin{array}{lcl}
[\sigma](\mathsf{dia}(E)) & = & \mathsf{dia}([\sigma]E) \\
[\sigma]\langle\tau, M\rangle & = & \langle[\sigma]\tau, [\sigma]M\rangle \\
[\sigma](\mathsf{letdia}(M, \langle\Psi, x\rangle.\, E)) & = & \mathsf{letdia}([\sigma]M, \langle\Psi, x\rangle.\, E) \\
[\sigma](\mathsf{letbox}(M, u.\, E)) & = & \mathsf{letbox}([\sigma]M, u.\, [\sigma]E)
\end{array}$$

Observe that in the letdia clause of the definition, the substitution σ is not applied to E; the free variables of E are all bound by Ψ, x, and thus cannot be influenced by σ.

Just as Pfenning and Davies [2001], we require a new substitution operation $\langle\!\langle F/\langle\Psi, x\rangle\rangle\!\rangle E$, where F and E are expressions witnessing relativized possibility. If $F : A\ \mathsf{poss}\langle\Psi\rangle$ and $\Psi, x{:}A\,\mathsf{true} \vdash E : B\ \mathsf{poss}\langle\Theta\rangle$ then $\langle\!\langle F/\langle\Psi, x\rangle\rangle\!\rangle E$ is a witness of relativized possibility $B\ \mathsf{poss}\langle\Theta\rangle$. The operation is defined by induction on the structure of F:

$$\begin{array}{rcl}
\langle\!\langle\langle\sigma, M\rangle/\langle\Psi, x\rangle\rangle\!\rangle E & = & [\sigma/\Psi, M/x]E \\
\langle\!\langle\mathsf{letdia}(M, \langle\Gamma, y\rangle.\, F')/\langle\Psi, x\rangle\rangle\!\rangle E & = & \mathsf{letdia}(M, \langle\Gamma, y\rangle.\, \langle\!\langle F'/\langle\Psi, x\rangle\rangle\!\rangle E) \\
\langle\!\langle\mathsf{letbox}(M, u.\, F')/\langle\Psi, x\rangle\rangle\!\rangle E & = & \mathsf{letbox}(M, u.\, \langle\!\langle F'/\langle\Psi, x\rangle\rangle\!\rangle E)
\end{array}$$

where we assume Γ, y and Ψ, x are variable disjoint. The local reduction and

expansion for contextual possibility can then be defined as:

$$\begin{array}{lll} \mathsf{letdia}(\mathsf{dia}(F), \langle\Psi, x\rangle.\, E) & \Longrightarrow_R & \langle\!\langle F/\langle\Psi, x\rangle\rangle\!\rangle E \\ \mathsf{letbox}(\mathsf{box}(\Psi.\, M), u.\, E) & \Longrightarrow_R & [\![\Psi.M/u]\!]E \\ M : \langle\Psi\rangle A & \Longrightarrow_E & \mathsf{dia}(\mathsf{letdia}(M, \langle\Psi, x\rangle.\, \langle \mathsf{id}_\Psi, x\rangle)) \end{array}$$

We currently do not have any convincing applications of contextual possibility, although the relationship to monadic programming [Pfenning and Davies 2001], mobile computation [Moody 2003], and, more generally, interactions of programs with their environment [Nanevski 2004] provide some avenues for further investigation.

9.2 Dependent necessity

Another direction for exploration is the dependent typing for the modal operator $\Box$ and its indexed variant. Currently, $\Box$ is defined only for simple types. In the dependent case we have a somewhat simplified operator $\Pi u{::}A[\Psi].B$, which internalizes the dependence on the meta-variable u, but does not internalize the validity judgment itself. As we explained in Section 5, this preserves the existence and properties of canonical forms of the underlying logical framework. However, if we are not concerned with canonical forms, the dependently typed rules for $\Box$ may be formulated as follows.

$$\frac{\Delta; \Psi \vdash M : A}{\Delta; \Gamma \vdash \mathsf{box}(\Psi.\, M) : [\Psi]A}\ \Box\mathsf{I}$$

$$\frac{\begin{array}{c}\Delta; \Gamma, x{:}[\Psi]A \vdash C(x) : \mathsf{type} \qquad \Delta; \Gamma \vdash M : [\Psi]A \\ \Delta, u{::}A[\Psi]; \Gamma \vdash N : C(\mathsf{box}(\Psi.\, \mathsf{clo}(u, \mathsf{id}_\Psi)))\end{array}}{\Delta; \Gamma \vdash \mathsf{letbox}(M, u.\, N) : C(M)}\ \Box\mathsf{E}$$

A complete dependent type theory with Π, Σ, equality and the full set of contextual modal operators is likely to be a suitable framework for reasoning about staged programs. For example, in such a type theory we could declare a staged, dependently typed version of the exponentiation function from Section 6 as follows,

$$\mathsf{exp} : \Pi x{:}\mathsf{int}.\ \Sigma e{:}[b{:}\mathsf{int}]\mathsf{int}.\ \Pi n{:}\mathsf{int}.\ \mathsf{letbox}(e, u.\, \mathsf{clo}(u, n/b)) = n^x$$

Here, the scope of the Σ plays the role of a correctness proof for e. It should be possible to eliminate this correctness proof with standard program extraction techniques or proof irrelevance [Pfenning 2001] to obtain the function presented earlier of type `int -> [b:int]int`.

9.3 Internalizing explicit substitutions

A further extension that we intend to pursue concerns internalizing simultaneous substitutions and giving them a first class status. This would allow us to pass substitutions as function arguments, and to quantify over substitution variables. Such an extension may potentially have interesting applications, especially in the modularity of encoding of object theories in a logical framework and perhaps in capturing operations on contexts such as closure conversion for a functional lan-

guage in a more immediate way than possible in LF. The proper setting for such an extension might be the ∇-calculus [Schürmann et al. 2005].

10. CONCLUSION

In this paper we present an intuitionistic contextual modal logic and its type theory (in both simple and dependently typed setting), as a suitable foundation for formalizing contexts in theorem proving and functional and logic programming. From the logical standpoint, our system is a relativized version of an intuitionistic modal logic S4. From the computational standpoint, it is a logically justified calculus of explicit simultaneous substitutions, with very pleasing theoretical properties.

As we outlined in the section on related work, various contextual lambda calculi have been proposed before, and various modal logics of contexts have been proposed before. However, we believe that we are the first to unify the two disparate views and application domains, and provide a type theoretic treatment for both.

We illustrate our logic and its type theory by applying it to two well-known problems in the areas of staged computation and logical frameworks.

In functional staged computation, one of the challenges has been to design a type system that can differentiate between code that is manipulated as data (object code), and code that performs the manipulation (meta code). Such a type system should satisfy two further requirements: (1) it should be possible to coerce object code with no free variables into meta code and execute it when required, and (2) it should be possible to compose object code with free variables into larger object code, while imposing capture of free variables. The two requirements are difficult to reconcile because the type system has to recognize when object code is closed in order to provide for (1), but also when object code is not closed and what its free variables are in order to provide for (2). Our type system provides for both by using the modal type constructor $[\Psi]A$ to classify object code with free variables Ψ; when Ψ is empty, that object code can be converted into meta code and executed.

In logical frameworks, meta-variables are place-holders for as yet unknown terms, to be instantiated via unification or via proof search. Each meta-variable admits terms that may depend only on a particular subset of parameters, and a potential unifier that does not respect this restriction has to be rejected. Tracking the correspondence between the meta-variables and their respective sets of admissible parameters is obviously very important for the soundness of the logical framework and can also have significant consequences for the framework efficiency.

Our proposal for meta-variables centers on the dependently typed version of intuitionistic contextual modal logic, where meta-variables correspond to hypotheses of contextual validity. For example, the meta-variable $u{::}A\,\mathsf{valid}[\Psi]$ admits only terms of type A that depend on the parameters listed in the local context Ψ. The modal formulation explains and justifies the optimization strategies like grafting, lowering and raising, that have previously been employed in implementations of logical frameworks and theorem provers, but have only been justified algorithmically, rather than logically. We also believe that ICML and its type theory is the first proposal that uncovers the modal nature of explicit substitutions in logical frameworks.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library by visiting the following URL: `http://www.acm.org/pubs/citations/journals/tocl/2007-V-N/p1-URLend`.

ACKNOWLEDGMENTS

We would like to thank Rowan Davies and Kevin Watkins for discussions relating to the subject of this paper, and Yukiyoshi Kameyama for comments on an earlier draft.

REFERENCES

Abadi, M., Cardelli, L., Curien, P.-L., and Lèvy, J.-J. 1990. Explicit substitutions. In *Symposium on Principles of Programming Languages, POPL'90*. ACM, San Francisco, California, 31–46.

Alechina, N., Mendler, M., de Paiva, V., and Ritter, E. 2001. Categorical and Kripke semantics for Constructive S4 modal logic. In *International Workshop on Computer Science Logic, CSL'01*, L. Fribourg, Ed. Lecture Notes in Computer Science, vol. 2142. Springer, Paris, France, 292–307.

Attardi, G. and Simi, M. 1994. Proofs in context. In *Principles of Knowledge Representation and Reasoning*, J. Doyle, E. Sandewall, and P. Torasso, Eds. 16–26.

Attardi, G. and Simi, M. 1995. A formalization of viewpoints. *Fundamenta Informaticae 23,* 3, 149–173.

Bjørner, N. and Muñoz, C. 2000. Absolute explicit unification. In *International Conference on Rewriting Techniques and Applications, RTA'00*. LNCS, vol. 1833. Norwich, UK, 31–46.

Bognar, M. and de Vrijer, R. 2001. A calculus of lambda calculus context. *Journal of Automated Reasoning 27,* 1, 29–59.

Buvač, S. 1996. Quantificational logic of context. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. 600–606.

Buvač, S., Buvač, V., and Mason, I. 1995. Metamathematics of contexts. *Fundamenta Informaticae 23,* 3, 412–419.

Calcagno, C., Moggi, E., and Taha, W. 2004. ML-like inference for classifiers. In *European Symposium on Programming, ESOP'04*, D. Schmidt, Ed. Springer-Verlag LNCS 2986, Barcelona, Spain, 79–93.

Cartmell, J. 1986. Generalized algebraic theories and contextual categories. *Annals of Pure and Applied Logic 32*, 209–243.

Cervesato, I. and Pfenning, F. 2002. A linear logical framework. *Information & Computation 179,* 1 (Nov.), 19–75.

Dami, L. 1998. A lambda-calculus for dynamic binding. *Theoretical Computer Science 192,* 2, 201–231.

Davies, R. 1996. A temporal logic approach to binding-time analysis. In *Symposium on Logic in Computer Science, LICS'96*, E. Clarke, Ed. IEEE Computer Society Press, New Brunswick, New Jersey, 184–195.

Davies, R. and Pfenning, F. 2001. A modal analysis of staged computation. *Journal of the ACM 48,* 3 (May), 555–604.

de Groote, P. 2002. On the strong normalisation of intuitionistic natural deduction with permutation-conversions. *Information and Computation 178,* 2 (Nov.), 441–464.

de Paiva, V. 2003. Natural deduction and context as (constructive) modality. In *Modelling and Using Context*, P. Blackburn, C. Ghidini, R. M. Turner, and F. Giunchiglia, Eds. Lecture Notes in Artificial Inteligence, vol. 2680. Springer, 116–129.

Dowek, G., Hardin, T., and Kirchner, C. 1995. Higher-order unification via explicit substitutions. In *Symposium on Logic in Computer Science, LICS'95*, D. Kozen, Ed. IEEE Computer Society Press, San Diego, California, 366–374.

Dowek, G., Hardin, T., Kirchner, C., and Pfenning, F. 1996. Unification via explicit substitutions: The case of higher-order patterns. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, M. Maher, Ed. MIT Press, Bonn, Germany, 259–273.

Geuvers, H. and Jojgov, G. I. 2002. Open proofs and open terms: a basis for interactive logic. In *International Workshop on Computer Science Logic, CSL'02*, J. C. Bradfield, Ed. Lecture Notes in Computer Science (LNCS 2471). Springer, Edinburgh, Scotland, 537–552.

Giunchiglia, F. 1993. Contextual reasoning. *Epistemologica 16*, 345–364.

Giunchiglia, F. and Serafini, L. 1994. Multilanguage hierarchical logics, or: How to do without modal logics. *Artificial Intelligence 65,* 1, 29–70.

Harper, R., Honsell, F., and Plotkin, G. 1993. A framework for defining logics. *Journal of the Association for Computing Machinery 40,* 1 (January), 143–184.

Hashimoto, M. and Ohori, A. 2001. A typed context calculus. *Theoretical Computer Science 266,* 1–2, 249–272.

Hodas, J. and Miller, D. 1994. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation 110,* 2, 327–365.

Jojgov, G. I. 2003. Holes with binding power. In *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, H. Geuvers and F. Wiedijk, Eds. Lecture Notes in Computer Science (LNCS 2646). Springer, 162–181.

Kripke, S. 1959. A completeness theorem in modal logic. *Journal of Symbolic Logic 24*, 1–14.

Lee, S.-D. and Friedman, D. P. 1996. Enriching the lambda calculus with contexts: toward a theory of incremental program construction. In *International Conference on Functional Programming, ICFP'96*. 239–250.

Liang, C. and Nadathur, G. 2002. Tradeoffs in the intensional representation of lambda terms. In *International Conference on Rewriting Techniques and Applications, RTA'02*, S. Tison, Ed. Springer-Verlag LNCS 2378, Copenhagen, Denmark, 192–206.

Magnusson, L. 1995. The implementation of ALF – a proof editor based on Martin-Löf's monomorphic type theory with explicit substitutions. Ph.D. thesis, Chalmers University of Technology and Göteborg University.

Martin-Löf, P. 1996. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic 1,* 1, 11–60.

Mason, I. A. 1999. Computing with contexts. *Higher-Order and Symbolic Computation 12,* 2, 171–201.

McCarthy, J. 1987. Generality in artificial intelligence. *Communications of the ACM 30,* 12, 1030–1035.

McCarthy, J. 1993. Notes on formalizing contexts. In *International Joint Conference on Artificial Intelligence*. Chambery, France, 555–560.

Miller, D. 1991. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation 1,* 4, 497–536.

Montague, R. 1963. Syntactical treatment of modalities, with corollaries on reflexion principles and finite axiomatizability. *Acta Philosophica Fennica 16*, 153–167.

Moody, J. 2003. Modal logic as a basis for distributed computation. Tech. Rep. CMU-CS-03-194, Carnegie Mellon University. Oct.

Muñoz, C. 2001. Dependent types and explicit substitutions. *Mathematical Structures in Computer Science 11,* 1 (February), 91–129.

Murphy VII, T., Crary, K., Harper, R., and Pfenning, F. 2004. A symmetric modal lambda calculus for distributed computing. In *Symposium on Logic in Computer Science, LICS'04*, H. Ganzinger, Ed. Turku, Finland, 286–295.

Nanevski, A. 2003. From dynamic binding to state via modal possibility. In *International Conference on Principles and Practice of Declarative Programming, PPDP'03*. Uppsala, Sweden, 207–218.

Nanevski, A. 2004. Functional programming with names and necessity. Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

Nanevski, A. and Pfenning, F. 2005. Staged computation with names and necessity. *Journal of Functional Programming 15,* 6 (Nov.), 837–891.

Nanevski, A., Pientka, B., and Pfenning, F. 2003. A modal foundation for meta variables. In *Proceedings of MERλIN 2003*. Uppsala, Sweden.

Pfenning, F. 2000. Structural cut elimination I. Intuitionistic and classical logic. *Information and Computation 157,* 1/2 (Mar.), 84–141.

Pfenning, F. 2001. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Symposium on Logic in Computer Science, LICS'01*, J. Halpern, Ed. IEEE Computer Society Press, Boston, Massachusetts, 221–230.

Pfenning, F. and Davies, R. 2001. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science 11,* 4, 511–540.

Pfenning, F. and Schürmann, C. 1999. System description: Twelf - a meta-logical framework for deductive systems. In *International Conference on Automated Deduction, CADE'99*, H. Ganzinger, Ed. Lecture Notes in Artificial Inteligence, vol. 1632. Springer-Verlag, Trento, Italy, 202–206.

Pientka, B. 2003. Tabled higher-order logic programming. Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

Pientka, B. and Pfennning, F. 2003. Optimizing higher-order pattern unification. In *International Conference on Automated Deduction, CADE'03*, F. Baader, Ed. Lecture Notes in Computer Science (LNAI 2741). Springer, Miami, Florida, 473–487.

Sato, M., Sakurai, T., and Burstall, R. 2001. Explicit environments. *Fundamenta Informaticae 45,* 1-2, 79–115.

Sato, M., Sakurai, T., and Kameyama, Y. 2002. A simply typed context calculus with first-class environments. *Journal of Functional and Logic Programming 2002,* 4 (March).

Sato, M., Sakurai, T., Kameyama, Y., and Igarashi, A. 2003. Calculi of meta-variables. In *International Workshop on Computer Science Logic, CSL'03*, M. Baaz and J. A. Makowsky, Eds. Lecture Notes in Computer Science (LNCS 2803). Springer, Vienna, Austria, 484–497.

Schürmann, C., Poswolsky, A., and Sarnat, J. 2005. The ∇-calculus: Functional programming with higher-order encodings. In *International Conference on Typed Lambda Calculus and Applications, TLCA'05*, P. Urzyczyn, Ed. Springer-Verlag LNCS 3461, Nara, Japan, 339–353.

Serafini, L. and Bouquet, P. 2004. Comparing formal theories of context in AI. *Artificial Intelligence 155,* 1–2, 41–67.

Serafini, L. and Giunchiglia, F. 2002. ML system: A proof theory for contexts. *Journal of Logic, Language and Information 11,* 4, 471–518.

Simpson, A. K. 1994. The proof theory and semantics of intuitionistic modal logic. Ph.D. thesis, University of Edinburgh.

Strecker, M. 1999. Construction and deduction in type theories. Ph.D. thesis, Universität Ulm.

Taha, W. and Nielsen, M. F. 2003. Environment classifiers. In *Symposium on Principles of Programming Languages, POPL'03*, G. Morrisett, Ed. ACM Press, New Orleans, Louisiana, 26–37.

Taha, W. and Sheard, T. 1997. Multi-stage programming with explicit annotations. In *Workshop on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*. Amsterdam, 203–217.

Thomason, R. H. 1999. Type theoretic foundations for context, part 1: Contexts as complex type-theoretic objects. In *Proceedings of the Second International and Interdisciplinary Conference on Modeling and Using Contexts, CONTEXT'99*, P. Bouquet, L. Serafini, P. Brézillon, M. Benerecetti, and F. Castellani, Eds. Trento, Italy, 352–374.

Thomason, R. H. 2003. Dynamic contextual intensional logic: Logical foundations and an application. In *Modeling and Using Context*, P. Blackburn, C. Ghidini, R. M. Turner, and F. Giunchiglia, Eds. Lecture Notes in Artificial Inteligence, vol. 2680. Springer, 328–341.

Watkins, K., Cervesato, I., Pfenning, F., and Walker, D. 2002. A concurrent logical framework I: Judgments and properties. Tech. Rep. CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University. Revised May 2003.

Weyhrauch, R. W. 1979. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence 13,* 1–2, 133–170.

Wickline, P., Lee, P., and Pfenning, F. 1998. Run-time code generation and Modal-ML. In *Conference on Programming Language Design and Implementation, PLDI'98*, K. D. Cooper, Ed. ACM Press, Montreal, Canada, 224–235.

Wickline, P., Lee, P., Pfenning, F., and Davies, R. 1998. Modal types as staging specifications for run-time code generation. *ACM Computing Surveys 30,* 3es.

Received September 2005; revised February 2007; accepted February 2007

This document is the online-only appendix to:

Contextual Modal Type Theory

ALEKSANDAR NANEVSKI
Harvard University
and
FRANK PFENNING
Carnegie Mellon University
and
BRIGITTE PIENTKA
McGill University

ACM Transactions on Computational Logic, Vol. V, No. N, March 2007, Pages 1–48.

A. TYPE SYSTEM OF THE DEPENDENT CMTT

Judgments.

$\vdash \Sigma$ sig	Σ is a valid signature.
$\vdash_\Sigma \Delta$ mctx	Δ is a valid modal context.
$\Delta \vdash_\Sigma \Gamma$ ctx	Γ is a valid context.
$\Delta;\Gamma \vdash_\Sigma K \Leftarrow$ kind	K is a valid kind.
$\Delta;\Gamma \vdash_\Sigma P \Rightarrow K$	atomic family P infers kind K.
$\Delta;\Gamma \vdash_\Sigma A \Leftarrow$ type	A is a valid normal type.
$\Delta;\Gamma \vdash_\Sigma R \Rightarrow A$	atomic object R infers type A.
$\Delta;\Gamma \vdash_\Sigma M \Leftarrow A$	normal object M checks against type A.
$\Delta;\Gamma \vdash_\Sigma \sigma \Leftarrow \Psi$	substitution σ checks against context Ψ.

Presuppositions. In all judgments but the validity of signatures ($\vdash \Sigma$ sig), we assume that the signature Σ is valid. In all judgments but the validity of modal contexts ($\vdash_\Sigma \Delta$ mctx) we assume the modal context Δ is valid. In all judgments but the validity of contexts ($\Delta \vdash_\Sigma \Gamma$ ctx) we assume the context Γ is valid in the modal context to its left. In the checking judgments $M \Leftarrow A$ and $\sigma \Leftarrow \Psi$ we assume that A and Ψ are given as valid types and contexts, respectively. In the synthesis judgments $P \Rightarrow K$ and $R \Rightarrow A$ we establish (rather than assuming) that K and A are valid. These presuppositions will be satisfied if rules are read from conclusion to premise, and the premises are read from left to right. We use explicit equality checks where a given input has to be compared with the output of another judgment.

Signature formation.

$$\frac{}{\vdash \cdot\ \mathsf{sig}} \qquad \frac{\vdash \Sigma\ \mathsf{sig} \quad \cdot;\cdot \vdash_\Sigma K \Leftarrow \mathsf{kind}}{\vdash (\Sigma, a{:}K)\ \mathsf{sig}} \qquad \frac{\vdash \Sigma\ \mathsf{sig} \quad \cdot;\cdot \vdash_\Sigma A \Leftarrow \mathsf{type}}{\vdash (\Sigma, c{:}A)\ \mathsf{sig}}$$

In the subsequent rules, we omit the subscript Σ from the judgments since it is the same in all premises and conclusion.

Modal context formation.

$$\frac{}{\vdash \cdot\ \mathsf{mctx}} \qquad \frac{\vdash \Delta\ \mathsf{mctx} \quad \Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type}}{\vdash (\Delta, u{::}A[\Psi])\ \mathsf{mctx}}$$

Context formation.

$$\frac{}{\Delta \vdash \cdot\ \mathsf{ctx}} \qquad \frac{\Delta \vdash \Gamma\ \mathsf{ctx} \quad \Delta;\Gamma \vdash A \Leftarrow \mathsf{type}}{\Delta \vdash (\Gamma, x{:}A)\ \mathsf{ctx}}$$

Kind formation.

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

$$\frac{\Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type} \quad \Delta, u{::}A[\Psi];\Gamma \vdash K \Leftarrow \mathsf{kind}}{\Delta;\Gamma \vdash \Pi u{::}A[\Psi].\, K \Leftarrow \mathsf{kind}}$$

Type formation.

$$\frac{a{:}K \in \Sigma}{\Delta;\Gamma \vdash a \Rightarrow K} \qquad \frac{\Delta;\Gamma \vdash P \Rightarrow \Pi x{:}A.K \quad \Delta;\Gamma \vdash N \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{app}(P, N) \Rightarrow [N/x]^k_A(K)}$$

$$\frac{\Delta;\Gamma \vdash P \Rightarrow \Pi u{::}A[\Psi].K \quad \Delta;\Psi \vdash N \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{mapp}(P, \hat{\Psi}.N) \Rightarrow [\![\hat{\Psi}.N/u]\!]^k_{A[\Psi]}(K)}$$

$$\frac{\Delta;\Gamma \vdash P \Rightarrow K \quad K = \mathsf{type}}{\Delta;\Gamma \vdash P \Leftarrow \mathsf{type}} \qquad \frac{\Delta;\Gamma \vdash A \Leftarrow \mathsf{type} \quad \Delta;\Gamma, x{:}A \vdash B \Leftarrow \mathsf{type}}{\Delta;\Gamma \vdash \Pi x{:}A.\, B \Leftarrow \mathsf{type}}$$

$$\frac{\Delta \vdash \Psi\ \mathsf{ctx} \quad \Delta;\Psi \vdash A \Leftarrow \mathsf{type} \quad \Delta, u{::}A[\Psi];\Gamma \vdash B \Leftarrow \mathsf{type}}{\Delta;\Gamma \vdash \Pi u{::}A[\Psi].\, B \Leftarrow \mathsf{type}}$$

Objects.

$$\frac{c{:}A \in \Sigma}{\Delta;\Gamma \vdash c \Rightarrow A} \qquad \frac{}{\Delta;\Gamma, x{:}A, \Gamma' \vdash x \Rightarrow A}$$

$$\frac{\Delta, u{::}A[\Psi], \Delta';\Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}A[\Psi], \Delta';\Gamma \vdash \mathsf{clo}(u,\sigma) \Rightarrow [\sigma]^a_\Psi(A)}$$

$$\frac{\Delta;\Gamma, x{:}A \vdash M \Leftarrow B}{\Delta;\Gamma \vdash \mathsf{lam}(x.\, M) \Leftarrow \Pi x{:}A.B} \qquad \frac{\Delta;\Gamma \vdash R \Rightarrow \Pi x{:}A.B \quad \Delta;\Gamma \vdash M \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{app}(R, M) \Rightarrow [M/x]^a_A(B)}$$

$$\frac{\Delta, u{::}A[\Psi];\Gamma \vdash M \Leftarrow B}{\Delta;\Gamma \vdash \mathsf{mlam}(u.\, M) \Leftarrow \Pi u{::}A[\Psi].B}$$

$$\frac{\Delta;\Gamma \vdash R \Rightarrow \Pi u{::}A[\Psi].B \quad \Delta;\Psi \vdash M \Leftarrow A}{\Delta;\Gamma \vdash \mathsf{mapp}(R, \hat{\Psi}.M) \Rightarrow [\![\hat{\Psi}.M/u]\!]^a_{A[\Psi]}(B)}$$

$$\frac{\Delta;\Gamma \vdash R \Rightarrow P' \quad P' = P}{\Delta;\Gamma \vdash R \Leftarrow P}$$

Substitutions.

$$\frac{}{\Delta;\Gamma \vdash (\cdot) \Leftarrow (\cdot)} \qquad \frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash M \Leftarrow [\sigma]^a_\Psi(A)}{\Delta;\Gamma \vdash (\sigma, M/x) \Leftarrow (\Psi, x{:}A)}$$

$$\frac{\Delta;\Gamma \vdash \sigma \Leftarrow \Psi \quad \Delta;\Gamma \vdash R \Rightarrow A' \quad A' = [\sigma]^a_\Psi(A)}{\Delta;\Gamma \vdash (\sigma, R/\!\!/x) \Leftarrow (\Psi, x{:}A)}$$

B. DEFINITION OF HEREDITARY SUBSTITUTIONS

B.1 Input and Output of Hereditary Substitutions

$$
\begin{array}{lcll}
[M/x]^n_\alpha(N) & = & N' & \\
[M/x]^r_\alpha(R) & = & R' & \text{or} \quad M' : \alpha' \\
[M/x]^s_\alpha(\tau) & = & \tau' & \\
[M/x]^a_\alpha(B) & = & B' & \\
[M/x]^p_\alpha(P) & = & P' & \\
[M/x]^k_\alpha(K) & = & K' & \\
[M/x]^\gamma_\alpha(\Gamma) & = & \Gamma' &
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) & = & N' & \\
[\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R) & = & R' & \text{or} \quad M' : \alpha' \\
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) & = & \tau' & \\
[\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(B) & = & B' & \\
[\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(P) & = & P' & \\
[\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(K) & = & K' & \\
[\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma) & = & \Gamma' & \\
[\![\hat{\Psi}.M/u]\!]^\delta_{\alpha[\psi]}(\Delta) & = & \Delta' &
\end{array}
$$

$$
\begin{array}{lcll}
[\sigma]^n_\psi(N) & = & N' & \\
[\sigma]^r_\psi(R) & = & R' & \text{or} \quad M' : \alpha' \\
[\sigma]^s_\psi(\tau) & = & \tau' & \\
[\sigma]^a_\psi(B) & = & B' & \\
[\sigma]^p_\psi(P) & = & P' &
\end{array}
$$

B.2 Hereditary Substitution for a Variable

$$
\begin{array}{lcll}
[M/x]^r_\alpha(x) & = & M : \alpha & \\
[M/x]^r_\alpha(y) & = & y & \text{if } y \neq x \\
[M/x]^r_\alpha(\mathsf{app}(R, N)) & = & \mathsf{app}(R', N') & \text{where } R' = [M/x]^r_\alpha(R) \\
 & & & \text{and } N' = [M/x]^n_\alpha(N) \\
[M/x]^r_\alpha(\mathsf{app}(R, N)) & = & M_2 : \alpha_2 & \text{if } [M/x]^r_\alpha(R) = \mathsf{lam}(y.\, M') : \alpha_1 \to \alpha_2, \\
 & & & \text{where } \alpha_1 \to \alpha_2 \leq \alpha \text{ and } N' = [M/x]^n_\alpha(N) \\
 & & & \text{and } M_2 = [N'/y]^n_{\alpha_1}(M') \\
[M/x]^r_\alpha(\mathsf{clo}(v, \tau)) & = & \mathsf{clo}(v, \tau') & \text{where } \tau' = [M/x]^s_\alpha(\tau) \\
[M/x]^r_\alpha(\mathsf{mapp}(R, \hat{\Gamma}.N)) & = & \mathsf{mapp}(R', \hat{\Gamma}.N) & \text{where } R' = [M/x]^r_\alpha(R) \\
[M/x]^r_\alpha(\mathsf{mapp}(R, \hat{\Gamma}.N)) & = & M_2 : \alpha_2 & \text{if } [M/x]^r_\alpha(R) = \mathsf{mlam}(u.\, M') : \alpha_1[\gamma] \Rightarrow \alpha_2 \\
 & & & \text{and } \alpha_1[\gamma] \Rightarrow \alpha_2 \leq \alpha \text{ and } \hat{\gamma} = \hat{\Gamma} \text{ and } M_2 = \\
 & & & [\![\hat{\Gamma}.N/u]\!]^n_{\alpha_1[\gamma]}(M') \\
[M/x]^r_\alpha(R) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^n_\alpha(\mathsf{lam}(y.\, N)) & = & \mathsf{lam}(y.\, N') & \text{where } N' = [M/x]^n_\alpha(N) \\
 & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
[M/x]^n_\alpha(\mathsf{mlam}(u.\, N)) & = & \mathsf{mlam}(u.\, N') & \text{where } N' = [M/x]^n_\alpha(N) \\
 & & & \text{choosing } u \notin \mathsf{FMV}(M) \\
[M/x]^n_\alpha(R) & = & M' & \text{if } [M/x]^r_\alpha(R) = M' : \alpha' \\
[M/x]^n_\alpha(R) & = & R' & \text{if } [M/x]^r_\alpha(R) = R' \\
[M/x]^n_\alpha(N) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^s_\alpha(\cdot) & = & \cdot & \\
[M/x]^s_\alpha(\tau, N/y) & = & \tau', N'/y & \text{where } \tau' = [M/x]^s_\alpha(\tau) \\
 & & & \text{and } N' = [M/x]^n_\alpha(N) \\
[M/x]^s_\alpha(\tau, R/\!\!/y) & = & \tau', R'/\!\!/y & \text{if } [M/x]^r_\alpha(R) = R' \text{ and } \tau' = [M/x]^s_\alpha(\tau) \\
[M/x]^s_\alpha(\tau, R/\!\!/y) & = & \tau', M'/y & \text{if } [M/x]^r_\alpha(R) = M' : \alpha' \\
 & & & \text{and } \tau' = [M/x]^s_\alpha(\tau) \\
[M/x]^s_\alpha(\tau) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^p_\alpha(a) & = & a & \\
[M/x]^p_\alpha(\mathsf{app}(P, N)) & = & \mathsf{app}(P', N') & \text{where } P' = [M/x]^p_\alpha(P) \\
 & & & \text{and } N' = [M/x]^n_\alpha(N) \\
[M/x]^p_\alpha(\mathsf{mapp}(P, \hat{\Gamma}.N)) & = & \mathsf{mapp}(P', \hat{\Gamma}.N) & \text{where } P' = [M/x]^p_\alpha(P) \\
[M/x]^p_\alpha(P) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^a_\alpha(P) & = & P' & \text{where } P' = [M/x]^p_\alpha(P) \\
[M/x]^a_\alpha(\Pi y{:}A.\, B) & = & \Pi y{:}A'.\, B' & \text{if } A' = [M/x]^a_\alpha(A) \text{ and } B' = [M/x]^a_\alpha(B) \\
 & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
[M/x]^a_\alpha(\Pi u{::}A[\Psi].\, B) & = & \Pi u{::}A[\Psi].\, B' & \text{if } B' = [M/x]^a_\alpha(B) \\
 & & & \text{choosing } u \notin \mathsf{FMV}(M) \\
[M/x]^a_\alpha(A) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^k_\alpha(\mathsf{type}) & = & \mathsf{type} & \\
[M/x]^k_\alpha(\Pi y{:}A.\, K) & = & \Pi y{:}A'.\, K' & \text{if } A' = [M/x]^a_\alpha(A) \\
 & & & \text{and } K' = [M/x]^k_\alpha(K) \\
 & & & \text{choosing } y \notin \mathsf{FV}(M) \text{ and } y \neq x \\
[M/x]^k_\alpha(\Pi u{::}A[\Psi].\, K) & = & \Pi u{::}A[\Psi].\, K' & \text{if } K' = [M/x]^a_\alpha(K) \\
 & & & \text{choosing } u \notin \mathsf{FMV}(M) \\
[M/x]^k_\alpha(K) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[M/x]^\gamma_\alpha(\cdot) & = & \cdot & \\
[M/x]^\gamma_\alpha(\Gamma, y{:}A) & = & \Gamma', y{:}A' & \text{if } \Gamma' = [M/x]^\gamma_\alpha(\Gamma) \\
 & & & \text{and } A' = [M/x]^a_\alpha(A) \\
[M/x]^\gamma_\alpha(\Gamma) & & \text{fails} & \text{otherwise}
\end{array}
$$

B.3 Hereditary Substitution for a Meta-Variable

$$
\begin{array}{lcll}
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(x) & = & x & \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{app}(R,N)) & = & \mathsf{app}(R',N') & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = R' \\
 & & & \text{with } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{app}(R,N)) & = & M_2 : \alpha_2 & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = \mathsf{lam}(y.\, M') : \\
 & & & \alpha_1 \to \alpha_2 \text{ for } \alpha_1 \to \alpha_2 \leq \alpha[\psi] \\
 & & & \text{and } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
 & & & \text{and } M_2 = [N'/y]^n_{\alpha_1}(M') \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{clo}(u,\tau)) & = & M' : \alpha & \text{where } \tau' = [\![\hat\Psi.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
 & & & \text{and } M' = [\tau']^n_\psi(M) \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{clo}(v,\tau)) & = & \mathsf{clo}(v,\tau') & \text{if } v \neq u \text{ with } \tau' = [\![\hat\Psi.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{mapp}(R,\hat\Gamma.N)) & = & \mathsf{mapp}(R',\hat\Gamma.N') & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = R' \\
 & & & \text{and } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(\mathsf{mapp}(R,\hat\Gamma.N)) & = & M_2 : \alpha_2 & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = \mathsf{mlam}(v.\, M') : \\
 & & & \alpha_1[\gamma] \Rightarrow \alpha_2 \text{ where } \alpha_1[\gamma] \Rightarrow \alpha_2 \leq \alpha[\psi] \\
 & & & \text{and } \hat\gamma = \hat\Gamma \\
 & & & \text{and } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
 & & & \text{and } M_2 = [\![\hat\Gamma.N'/v]\!]^n_{\alpha_1[\gamma]}(M') \\
[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(\mathsf{lam}(y.\, N)) & = & \mathsf{lam}(y.\, N') & \text{where } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(\mathsf{mlam}(v.\, N)) & = & \mathsf{mlam}(v.\, N') & \text{where } N' = [\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) \\
 & & & \text{choosing } v \notin \mathsf{FMV}(M) \text{ and } v \neq u \\
[\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(R) & = & R' & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = R' \\
[\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(R) & = & M' & \text{if } [\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = M' : \alpha' \\
[\![\hat\Psi.M/u]\!]^n_{\alpha[\psi]}(N) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\cdot) & = & \cdot & \\
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau, N/y) & = & \tau', N'/y & \text{where } \tau' = [\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
 & & & \text{and } N' = [\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau, R/\!\!/y) & = & \tau', R'/\!\!/y & \text{if } [\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R) = R' \\
 & & & \text{with } \tau' = [\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau, R/\!\!/y) & = & \tau', M'/y & \text{if } [\![\hat{\Psi}.M/u]\!]^r_{\alpha[\psi]}(R) = M' : \alpha' \\
 & & & \text{with } \tau' = [\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) \\
[\![\hat{\Psi}.M/u]\!]^s_{\alpha[\psi]}(\tau) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(a) & = & a & \\
[\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(\mathsf{app}(P, N)) & = & \mathsf{app}(P', N') & \text{if } P' = [\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(P) \\
 & & & \text{and } N' = [\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(\mathsf{mapp}(P, \hat{\Gamma}.N)) & = & \mathsf{mapp}(P', \hat{\Gamma}.N') & \text{if } P' = [\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(P) \\
 & & & \text{and } N' = [\![\hat{\Psi}.M/u]\!]^n_{\alpha[\psi]}(N) \\
[\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(P) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(P) & = & P' & \text{where } P' = [\![\hat{\Psi}.M/u]\!]^p_{\alpha[\psi]}(P) \\
[\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(\Pi x{:}A.\, B) & = & \Pi x{:}A'.\, B' & \text{where } A' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) \\
 & & & \text{and } B' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(B) \\
[\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(\Pi v{::}A[\Gamma].\, B) & = & \Pi v{::}A'[\Gamma'].\, B' & \text{where } A' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) \\
 & & & \text{and } \Gamma' = [\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma) \\
 & & & \text{and } B' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(B) \\
 & & & \text{choosing } v \notin \mathsf{FMV}(M) \\
 & & & \text{and } v \neq u \\
[\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(\mathsf{type}) & = & \mathsf{type} & \\
[\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(\Pi x{:}A.\, K) & = & \Pi x{:}A'.\, K' & \text{where } A' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) \\
 & & & \text{and } K' = [\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(K) \\
[\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(\Pi v{::}A[\Gamma].\, K) & = & \Pi v{::}A'[\Gamma'].\, K' & \text{where } A' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) \\
 & & & \text{and } \Gamma' = [\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma) \\
 & & & \text{and } K' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(K) \\
 & & & \text{choosing } v \notin \mathsf{FMV}(M) \\
 & & & \text{and } v \neq u \\
[\![\hat{\Psi}.M/u]\!]^k_{\alpha[\psi]}(K) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\cdot) & = & \cdot & \\
[\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma, x{:}A) & = & \Gamma', x{:}A' & \text{where } \Gamma' = [\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma) \\
 & & & \text{and } A' = [\![\hat{\Psi}.M/u]\!]^a_{\alpha[\psi]}(A) \\
[\![\hat{\Psi}.M/u]\!]^\gamma_{\alpha[\psi]}(\Gamma) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\![\hat{\Psi}.M/u]\!]^{\delta}_{\alpha[\psi]}(\cdot) & = & \cdot & \\
[\![\hat{\Psi}.M/u]\!]^{\delta}_{\alpha[\psi]}(\Delta, v{::}A[\Gamma]) & = & \Delta', v{::}A'[\Gamma'] & \text{where } \Delta' = [\![\hat{\Psi}.M/y]\!]^{\delta}_{\alpha[\psi]}(\Delta) \\
 & & & \text{and } A' = [\![\hat{\Psi}.M/y]\!]^{a}_{\alpha[\psi]}(A) \\
 & & & \text{and } \Gamma' = [\![\hat{\Psi}.M/y]\!]^{\gamma}_{\alpha[\psi]}(\Gamma) \\
 & & & \text{choosing } v \neq u \\
[\![\hat{\Psi}.M/u]\!]^{\delta}_{\alpha[\psi]}(\Delta) & & \text{fails} & \text{otherwise}
\end{array}
$$

B.4 Hereditary Simultaneous Substitution

$$
\begin{array}{lcll}
[\sigma]^{r}_{\psi}(x) & = & M : \alpha & \text{if } M/x \in \sigma/\psi \text{ and } x{:}\alpha \in \psi \\
[\sigma]^{r}_{\psi}(x) & = & R & \text{if } R/\!\!/x \in \sigma/\psi \\
[\sigma]^{r}_{\psi}(\mathsf{app}(R, N)) & = & \mathsf{app}(R', N') & \text{where } [\sigma]^{r}_{\psi}(R) = R' \text{ and } [\sigma]^{n}_{\psi}(N) = N' \\
[\sigma]^{r}_{\psi}(\mathsf{app}(R, N)) & = & M_2 : \alpha_2 & \text{if } [\sigma]^{r}_{\psi}(R) = \mathsf{lam}(y.\, M') : \alpha_1 \to \alpha_2 \text{ for } \alpha_1 \to \alpha_2 \leq \psi \text{ with } N' = [\sigma]^{n}_{\psi}(N) \\
 & & & \text{and } M_2 = [N'/y]^{n}_{\alpha_1}(M') \\
[\sigma]^{r}_{\psi}(\mathsf{clo}(v, \tau)) & = & \mathsf{clo}(v, \tau') & \text{where } \tau' = [\sigma]^{s}_{\psi}(\tau) \\
[\sigma]^{r}_{\psi}(\mathsf{mapp}(R, \hat{\Gamma}.N)) & = & \mathsf{mapp}(R', \hat{\Gamma}.N) & \text{if } [\sigma]^{r}_{\psi}(R) = R' \\
[\sigma]^{r}_{\psi}(\mathsf{mapp}(R, \hat{\Gamma}.N)) & = & M_2 : \alpha_2 & \text{if } [\sigma]^{r}_{\psi}(R) = \mathsf{mlam}(u.\, M) : \alpha_1[\gamma] \Rightarrow \alpha_2 \text{ for } \alpha_1[\gamma] \Rightarrow \alpha_2 \leq \psi \text{ and } \hat{\gamma} = \hat{\Gamma} \text{ and } M_2 = [\![\hat{\Gamma}.N/u]\!]^{n}_{\alpha_1[\gamma]}(M) \\
[\sigma]^{r}_{\psi}(R) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\sigma]^{n}_{\psi}(\mathsf{lam}(y.\, N)) & = & \mathsf{lam}(y.\, N') & \text{where } N' = [\sigma, y/\!\!/y]^{n}_{\psi, y:\_}(N) \\
 & & & \text{choosing } y \notin \mathsf{FV}(\sigma), \mathsf{dom}(\sigma) \\
[\sigma]^{n}_{\psi}(\mathsf{mlam}(u.\, N)) & = & \mathsf{mlam}(u.\, N') & \text{where } N' = [\sigma]^{n}_{\psi}(N), \text{ choosing } u \notin \mathsf{FMV}(\sigma) \\
[\sigma]^{n}_{\psi}(R) & = & M' & \text{if } [\sigma]^{r}_{\psi}(R) = M' : \alpha' \\
[\sigma]^{n}_{\psi}(R) & = & R' & \text{if } [\sigma]^{r}_{\psi}(R) = R' \\
[\sigma]^{n}_{\psi}(N) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\sigma]^{s}_{\psi}(\cdot) & = & \cdot & \\
[\sigma]^{s}_{\psi}(\tau, N/y) & = & \tau', N'/y & \text{where } \tau' = [\sigma]^{s}_{\psi}(\tau) \text{ and } N' = [\sigma]^{n}_{\psi}(N) \\
[\sigma]^{s}_{\psi}(\tau, R/\!\!/y) & = & \tau', R'/\!\!/y & \text{if } [\sigma]^{r}_{\psi}(R) = R' \text{ with } \tau' = [\sigma]^{s}_{\psi}(\tau) \\
[\sigma]^{s}_{\psi}(\tau, R/\!\!/y) & = & \tau', M'/y & \text{if } [\sigma]^{r}_{\psi}(R) = M' : \alpha' \text{ with } \tau' = [\sigma]^{s}_{\psi}(\tau) \\
[\sigma]^{s}_{\psi}(\tau) & & \text{fails} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
[\sigma]^{p}_{\psi}(a) & = & a & \\
[\sigma]^{p}_{\psi}(\mathsf{app}(P, N)) & = & \mathsf{app}(P', N') & \text{where } P' = [\sigma]^{p}_{\psi}(P) \text{ and } N' = [\sigma]^{n}_{\psi}(N) \\
[\sigma]^{p}_{\psi}(\mathsf{mapp}(P, \hat{\Gamma}.N)) & = & \mathsf{mapp}(P', \hat{\Gamma}.N) & \text{where } P' = [\sigma]^{p}_{\psi}(P) \\
[\sigma]^{p}_{\psi}(P) & & \text{fails} & \text{otherwise}
\end{array}
$$

$[\sigma]^a_\psi(P)$	$=$	P'	where $P' = [\sigma]^p_\psi(P)$
$[\sigma]^a_\psi(\Pi x{:}A.\, B)$	$=$	$\Pi x{:}A'.\, B'$	if $A' = [\sigma]^a_\psi(A)$ and $B' = [\sigma, x/\!\!/x]^a_{\psi,x:\_}(B)$ choosing $x \notin \mathsf{FV}(\sigma), \mathsf{dom}(\sigma)$
$[\sigma]^a_\psi(\Pi u{::}A[\Gamma].\, B)$	$=$	$\Pi u{::}A[\Gamma].\, B'$	if $B' = [\sigma]^a_\psi(B)$ choosing $u \notin \mathsf{FMV}(\sigma)$
$[\sigma]^a_\psi(A)$		fails	otherwise

C. SUBSTITUTION PRINCIPLES

We begin by defining the *head* of an atomic term as the variable or meta-variable at its beginning.

$$\begin{aligned} \mathsf{head}(x) &= x \\ \mathsf{head}(\mathsf{clo}(u,\tau)) &= u \\ \mathsf{head}(\mathsf{app}(R,M)) &= \mathsf{head}(R) \\ \mathsf{head}(\mathsf{mapp}(R,\Psi.M)) &= \mathsf{head}(R) \end{aligned}$$

Whether the result of a hereditary substitution into an atomic term R is an atomic term R' or a normal term $M' : \alpha'$ depends solely on the head variable of R. This property is needed in several subsequent proofs, so we state it formally.

Lemma C.1 Hereditary Substitutions and Heads.

(1) *If* $[M/x]^r_\alpha(R)$ *exists, then*
  (a) $[M/x]^r_\alpha(R) = R'$ *is atomic iff* $\mathsf{head}(R) \neq x$.
  (b) $[M/x]^r_\alpha(R) = M' : \alpha'$ *is normal iff* $\mathsf{head}(R) = x$

(2) *If* $[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R)$ *exists, then*
  (a) $[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = R'$ *is atomic iff* $\mathsf{head}(R) \neq u$
  (b) $[\![\hat\Psi.M/u]\!]^r_{\alpha[\psi]}(R) = M' : \alpha'$ *is normal iff* $\mathsf{head}(R) = u$

(3) *If* $[\sigma]^r_\psi(R)$ *exists, then*
  (a) $[\sigma]^r_\psi(R) = R'$ *is atomic iff* $\mathsf{head}(R) = x$ *and* $R''/\!\!/x \in \sigma/\psi$.
  (b) $[\sigma]^r_\psi(R) = M' : \alpha'$ *is normal iff* $\mathsf{head}(R) = x$ *and* $M''/x \in \sigma/\psi$.

Proof. By straightforward induction on the structure of R. □

Next we need to show that hereditary substitutions are defined and do not change the term if a variable does not appear in the term we substitute into.

Lemma C.2 Trivial Hereditary Substitutions. *Let T range over expressions of any syntactic category (i.e., kinds, atomic types, normal types, atomic terms, normal terms, substitutions, contexts and modal contexts), and let $* \in \{k, p, a, r, n, s, \gamma, \delta\}$, correspondingly.*

(1) *If* $x \notin \mathsf{FV}(T)$, *then* $[M/x]^*_\alpha(T) = T$.

(2) *If* $u \notin \mathsf{FMV}(T)$, *then* $[\![\hat\Psi.M/u]\!]^*_{\alpha[\psi]}(T) = T$.

(3) *If* $x \notin \mathsf{FV}(T)$, *then* $[\sigma, R/\!\!/x]^*_{\psi,x:\_}(T) = [\sigma, M/x]^*_{\psi,x:\alpha}(T) = [\sigma]^*_\psi(T)$.

Proof. By straightforward induction on the structure of T. □

When investigating properties of ordinary substitution, we usually have to verify that $[M/x]([N/y]O) = ([M/x]N/y)([M/x]O)$, assuming that y is not in the free variables of M. Similar principles apply to hereditary substitutions, although the fact that hereditary substitutions are partial operations on ill-typed terms makes their formulation more tedious.

LEMMA C.3 COMPOSITION OF HEREDITARY SUBSTITUTIONS. *Suppose that T ranges over expressions of any syntactic category (i.e., kinds, atomic types, normal types, atomic terms, normal terms, substitutions, contexts and modal contexts), and let $* \in \{k, p, a, r, n, s, \gamma, \delta\}$, correspondingly.*

(1) *If $y \notin \mathsf{FV}(M_0)$, and $[M_0/x]^*_\alpha(T) = T_0$, $[M_1/y]^*_\beta(T) = T_1$ and $[M_0/x]^n_\alpha(M_1)$ exist, then $[M_0/x]^*_\alpha(T_1) = [[M_0/x]^n_\alpha(M_1)/y]^*_\beta(T_0)$.*

(2) *If $v \notin \mathsf{FMV}(M_0)$, and $[M_0/x]^*_\alpha(T) = T_0$, $[\![\hat{\Psi}_1.M_1/v]\!]^*_{\beta[\psi_1]}(T) = T_1$, then $[M_0/x]^*_\alpha(T_1) = [\![\hat{\Psi}_1.M_1/v]\!]^*_{\beta[\psi_1]}(T_0)$.*

(3) *If $[\sigma_1]^*_{\psi_1}(T) = T_1$ and $[M_0/x]^s_\alpha(\sigma_1)$ exists, then $[M_0/x]^*_\alpha(T_1) = [[M_0/x]^s_\alpha(\sigma_1)]^*_{\psi_1}(T)$.*

(4) *If $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T) = T_0$ and $[M_1/y]^*_\beta(T) = T_1$ and $[\![\hat{\Psi}_0.M_0/u]\!]^n_{\alpha[\psi_0]}(M_1) = M_1'$ exist, then $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T_1) = [M_1'/y]^*_\beta(T_0)$.*

(5) *If $v \notin \mathsf{FMV}(M_0)$, and $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T) = T_0$ and $[\![\hat{\Psi}_1.M_1/v]\!]^*_{\beta[\psi_1]}(T) = T_1$ and $[\![\hat{\Psi}_0.M_0/u]\!]^n_{\alpha[\psi_0]}(M_1) = M_1'$ exist, then $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T_1) = [\![\hat{\Psi}_1.M_1'/v]\!]^*_{\beta[\psi_1]}(T_0)$.*

(6) *If $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T) = T_0$ and $[\sigma_1]^*_{\psi_1}(T) = T_1$ and $[\![\hat{\Psi}_0.M_0/u]\!]^s_{\alpha[\psi_0]}(\sigma_1) = \sigma_1'$ exist, then $[\![\hat{\Psi}_0.M_0/u]\!]^*_{\alpha[\psi_0]}(T_1) = [\sigma_1']^*_{\psi_1}(T_0)$.*

(7) *If $y \notin \mathsf{dom}(\sigma_0), \mathsf{FV}(\sigma_0)$, and $[\sigma_0, y/\!\!/y]^*_{\psi_0, y:\_}(T) = T_0$ and $[M_1/y]^*_\beta(T) = T_1$, and $[\sigma_0]^n_{\psi_0}(M)$ exists, then $[\sigma_0]^*_{\psi_0}(T_1) = [[\sigma_0]^n_{\psi_0}(M_1)/y]^*_\beta(T_0)$.*

(8) *If $v \notin \mathsf{FMV}(\sigma_0)$, and $[\sigma_0]^*_{\psi_0}(T) = T_0$ and $[\![\hat{\Psi}_1.M_1/v]\!]^*_{\beta[\psi_1]}(T) = T_1$, then $[\sigma_0]^*_{\psi_0}(T_1) = [\![\hat{\Psi}_1.M_1/v]\!]^*_{\beta[\psi_1]}(T_0)$.*

(9) *If $[\tau_1]^*_{\psi_1}(T)$ and $[\tau_0]^s_{\psi_0}(\tau_1)$ exist, then $[\tau_0]^*_{\psi_0}([\tau_1]^*_{\psi_1}(T)) = [[\tau_0]^s_{\psi_0}(\tau_1)]^*_{\psi_1}(T)$.*

PROOF. By nested induction, first on approximate types α, β and contexts ψ_0, ψ_1, and then on the structure of the expression being substituted into. □

Before we can state the hereditary substitution principles we have to overcome one further technical obstacle. When applying a simultaneous hereditary substitution to an abstraction, we do not have a type or approximate type for the abstracted variable available. For example,

$$[\sigma]^n_\psi(\mathsf{lam}(y.\, N)) = \mathsf{lam}(y.\, N') \qquad \text{where } N' = [\sigma, y/\!\!/y]^n_{\psi, y:\_}(N)$$

where we choose $y \notin \mathsf{FV}(\sigma), \mathsf{dom}(\sigma)$. That means that we can not maintain the invariant that $\psi = \Psi^-$ whenever we apply $[\sigma]_\psi$ for $\sigma \Leftarrow \Psi$. We say that ψ *underapproximates* Ψ with respect to σ if ψ differs from Ψ^- only in that some declarations $x{:}\alpha$ may be replaced by $x{:}\_$ if $R/\!\!/x$ is in σ. Fortunately, any underapproximation is still sufficient to guarantee termination and the desired hereditary substitution properties.

Now we are prepared to state the hereditary substitution principles. We assume that all contexts in given judgments are well-formed. We also exploit our prior convention that all dependencies in an index to a hereditary substitution are erased before the substitution is applied, writing, for example, $[M/x]^r_A(R)$ instead of $[M/x]^r_{A^-}(R)$.

THEOREM C.4 HEREDITARY SUBSTITUTION PRINCIPLES.
Assume all contexts in the judgments given below are well-formed.

(1) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A, \Gamma_1 \vdash R \Rightarrow C$ *and the context* $\Gamma_1' = [M/x]^\gamma_A(\Gamma_1)$ *exists and is well-formed (i.e.,* $\Delta \vdash \Gamma, \Gamma_1'$ ctx*), then* $[M/x]^r_A(R)$ *exists and the type* $C' = [M/x]^a_A(C)$ *exists and is well-formed (i.e.,* $\Delta;\Gamma,\Gamma_1' \vdash C' \Leftarrow \mathsf{type}$*) and*
   (a) *if* $[M/x]^r_A(R) = R'$ *is atomic, then* $\Delta;\Gamma,\Gamma_1' \vdash R' \Rightarrow C'$.
   (b) *if* $[M/x]^r_A(R) = M' : \alpha'$ *is normal, then* $\Delta;\Gamma,\Gamma_1' \vdash M' \Leftarrow C'$ *and* $\alpha' = C^-$.

(2) *If* $\Delta;\Gamma \vdash M \Leftarrow A$ *and* $\Delta;\Gamma, x{:}A, \Gamma_1 \vdash N \Leftarrow C$ *and the context* $\Gamma_1' = [M/x]^\gamma_A(\Gamma_1)$ *and type* $C' = [M/x]^a_A(C)$ *exist and are well-formed (i.e.,* $\Delta \vdash \Gamma, \Gamma_1'$ ctx *and* $\Delta;\Gamma,\Gamma_1' \vdash C' \Leftarrow \mathsf{type}$*), then* $\Delta;\Gamma,\Gamma_1' \vdash [M/x]^n_A(N) \Leftarrow C'$.

(3) *If* $\Delta;\Psi \vdash M \Leftarrow A$ *and* $\Delta, u{::}A[\Psi], \Delta_1;\Gamma \vdash R \Rightarrow C$*, and the modal context* $\Delta_1' = [\![\hat\Psi.M/u]\!]^\delta_{A[\Psi]}(\Delta_1)$ *and the context* $\Gamma' = [\![\hat\Psi.M/u]\!]^\gamma_{A[\Psi]}(\Gamma)$ *exist and are well-formed (i.e.,* $\vdash \Delta, \Delta_1'$ mctx*, and* $\Delta, \Delta_1' \vdash \Gamma'$ ctx*), then* $[\![\hat\Psi.M/u]\!]^r_{A[\Psi]}(R)$ *exists and the type* $C' = [\![\hat\Psi.M/u]\!]^a_{A[\Psi]}(C)$ *exists and is well-formed (i.e.* $\Delta, \Delta';\Gamma' \vdash C' \Leftarrow \mathsf{type}$*) and*
   (a) *if* $[\![\hat\Psi.M/u]\!]^r_{A[\Psi]}(R) = R'$ *is atomic, then* $\Delta, \Delta_1';\Gamma' \vdash R' \Rightarrow C'$
   (b) *if* $[\![\hat\Psi.M/u]\!]^r_{A[\Psi]}(R) = M' : \alpha'$ *is normal, then* $\Delta, \Delta_1';\Gamma' \vdash M' \Leftarrow C'$*, and* $\alpha' = C^-$.

(4) *If* $\Delta;\Psi \vdash M \Leftarrow A$ *and* $\Delta, u{::}A[\Psi], \Delta_1;\Gamma \vdash N \Leftarrow C$*, and the modal context* $\Delta_1' = [\![\hat\Psi.M/u]\!]^\delta_{A[\Psi]}(\Delta_1)$ *and the context* $\Gamma' = [\![\hat\Psi.M/u]\!]^\gamma_{A[\Psi]}(\Gamma)$ *and the type* $C' = [\![\hat\Psi.M/u]\!]^a_{A[\Psi]}(C)$ *exist and are well-formed (i.e.,* $\vdash \Delta, \Delta_1'$ mctx*, and* $\Delta, \Delta_1' \vdash \Gamma'$ ctx *and* $\Delta, \Delta_1';\Gamma' \vdash C' \Leftarrow \mathsf{type}$*), then* $\Delta, \Delta';\Gamma' \vdash [\![\hat\Psi.M/u]\!]^n_{A[\Psi]}(N) \Leftarrow C'$.

(5) *If* $\Delta;\Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta;\Psi \vdash R \Rightarrow C$ *and* ψ *is any underapproximation of* Ψ *with respect to* σ*, then* $[\sigma]^r_\psi(R)$ *exists and the type* $C' = [\sigma]^a_\psi(C)$ *exists and is well-formed (i.e.,* $\Delta;\Gamma \vdash C' \Leftarrow \mathsf{type}$*) and*
   (a) *if* $[\sigma]^r_\psi(R) = R'$ *is atomic, then* $\Delta;\Gamma \vdash R' \Rightarrow C'$
   (b) *if* $[\sigma]^r_\psi(R) = M' : \alpha'$ *is normal, then* $\Delta;\Gamma \vdash M' \Leftarrow C'$*, and* $\alpha' = C^-$.

(6) *If* $\Delta;\Gamma \vdash \sigma \Leftarrow \Psi$ *and* $\Delta;\Psi \vdash N \Leftarrow C$ *and* ψ *is any underapproximation of* Ψ *with respect to* σ *such that the type* $C' = [\sigma]^a_\psi(C)$ *exists and is well-formed (i.e.,* $\Delta;\Gamma \vdash C' \Leftarrow \mathsf{type}$*), then* $\Delta;\Gamma \vdash [\sigma]^n_\psi(N) \Leftarrow C'$.

PROOF. We generalize this property and then proceed by nested induction, first on the structure of the approximation A^- and underapproximation ψ to the index type A and index context Ψ, and then on the structure of the second derivation in each of the cases, using the lemmas on trivial hereditary substitutions and composition of hereditary substitutions (Lemmas C.2 and C.3). We also use that the erasure of a type is invariant under substitution (for example, $([M/x]^a_A(B))^- = B^-$) without explicitly stating this as lemma.

The generalization involves stating the similar principles for hereditary substitutions into other syntactic categories, that is, kinds, atomic types, normal types, contexts and modal contexts. We omit these cases here because hereditary substitutions into the mentioned syntactic categories are mostly compositional, and their substitution principles do not contribute any new insights. □