Towards a Dependent Modal Type Theory

Aleksandar Nanevski, Brigitte Pientka, Frank Pfenning
Carnegie Mellon University
Pittsburgh, PA 15213
{aleks,bp,fp}@cs.cmu.edu

June 20, 2003

Abstract

In this paper we present a dependently typed modal lambda calculus based on Constructive S4 modal logic as proposed by Pfenning and Davies. We consider notions of relative validity (relative to a context Ψ) and simultaneous possibility (simultaneous with a context Ψ). We internalize simultaneous possibility and provide a family of modal operators $\Diamond_\Psi$. In this paper, we do not fully internalize relative validity into a family of modal operators $\Box_\Psi$. Instead, we only allow universal quantification over valid variables. We also do not consider the equality judgment on objects and families. These two extensions are left for future work.

1 Introduction

Intuitionistic modal logic provides an expressive framework for specifying and analyzing different aspects of computation. The power of this approach comes from the modal operators, which allow the distinction between the strata of syntactic and semantics entities.

In functional programming it serves as a foundation for analyzing and explaining important implementation techniques and optimizations such as staged computation and run-time code generation [7, 16, 17, 2]. From a high-level simplified view, the modal types are seen as classifying closed *source* programs. This is in contrast to the non-modal types which classify *compiled* programs. Source programs may be bound to modal variables, composed into larger source programs, and also compiled and evaluated at run-time.

In automating the reasoning about programming languages, Despeyroux *et. all* [5] investigated the ideas of modal logic to preserve advantages of higher-order abstract syntax and at the same time allow inductive reasoning in this setting. Here the modal operator is used to separate recursive arguments from non-recursive arguments. This in turn facilitates a decomposition of the functions space into a non-parametric primitive recursive function space and a parametric function space making it possible to combine higher-order abstract syntax with inductive reasoning. The approach even scales to the dependently typed setting as shown in [4].

Recently, modal calculi have been extended to not only capture truth, validity and possibility, but also validity relative to a set Ψ of specified assumptions. This extension has several applications in functional programming and automated reasoning, which may differ in the exact nature of the set Ψ. For example, in staged computation and run-time code generation, modal variables are bound to source programs, and the set Ψ lists the free variables that these source expressions may contain [8]. With a similar motivation, the calculus of [10] interprets Ψ as a set of types. When Ψ is seen as a set of store locations that the modal expression can de-reference, then we obtain a calculus of dynamic binding [9].

In higher-order logic programming [11], relative validity has been used to characterize the existential variables and clearly separate them from bound variables. In this context, existential variables are viewed as modal variables of type A in a context Ψ. This allows the concise description of higher-order unification and explains a number of optimizations of its implementations [14].

Dual to the relative validity, we can also consider simultaneous possibility of a set of assumptions Ψ and a proposition A. This has already shown useful in a simply typed setting in [9], where this fragment formulates a calculus of state with second-class locations.

In this paper, we take a first step towards a general framework for a dependently typed modal calculus which allows both relative validity and simultaneous possibility. This combines and extends the previously mentioned proposals in [8, 9, 14]. We also introduce constructs for universal quantification over modal variables, and internalize simultaneous possibility by adding a family of modal operators $\Diamond_\Psi$. This generalizes the modal operator $\Diamond$ from constructive propositional modal logic S4 [12, 1]. We see our calculus as a first step toward a full-fledged modal type theory. Potential applications of this modal type theory lie for example in factoring and re-using sub-proofs, explaining notational definitions, interaction of constraints and higher-order logic programming, and induction principles with regular worlds for higher-order abstract syntax.

The paper is outlined as follows: We first give some background on extending modal logic with relative truth and validity, and introduce a dependently typed calculus based on it. Next, we present extensions with quantification over modal variables, and also an extension with the modal possibility operator. Finally, we summarize the central ideas and discuss future work.

2 Background

2.1 Motivation

Our development of the dependently typed modal calculus follows the methodology of Pfenning and Davies [12]. The key idea in [12] is to assign constructive explanations to modal operators by distinguishing between propositions that are true, propositions that are valid, and propositions that are possible. A proposition is valid if its truth does not depend on the truth of any other proposition. A proposition is possible if it is true or if its truth depends on other possible propositions. This leads to the basic hypothetical judgments

$$A_1 \; valid, \dots A_n \; valid; B_1 \; true, \dots, B_m \; true \vdash C \; true$$

and

$$A_1 \; valid, \dots A_n \; valid; B_1 \; true, \dots, B_m \; true \vdash C \; poss.$$

which are abbreviated as $\Delta; \Gamma \vdash C \; true$, and $\Delta; \Gamma \vdash C \; poss$.

Under the multiple-world interpretation of Constructive S4 modal logic, $C \; valid$ corresponds to $C \; true$ in *all* reachable worlds. This means $C \; true$ without any assumptions, except those that are assumed to be true in all worlds. On the other hand, $C \; poss$ corresponds to $C \; true$ in *some* reachable world, or more specifically

1. if $\Delta; \Gamma \vdash C \; true$ then $\Delta; \Gamma \vdash C \; poss$ (because the current world is assumed reachable), or
2. if $\Delta; \Gamma \vdash A \; poss$ and $\Delta; A \; true \vdash C \; poss$ then $\Delta; \Gamma \vdash C \; poss$

We can generalize this idea to also capture truth relative to a set of specified assumptions by writing $C \; valid \; \Psi$, where Ψ is the abbreviation for $C_1 \; true, \dots, C_n \; true$. In terms of the multiple world semantics, this means that C is true in any world where C_1 through C_n are all true and we say C is valid relative to the assumptions in Ψ. Hypotheses about relative validity are more complex now, so our general judgment form is

$$A_1 \; valid \; \Psi_1, \dots, A_n \; valid \; \Psi_n; B_1 \; true, \dots, B_m \; true \vdash C \; true.$$

If we have an assumption $A \; valid \; \Psi$ we can only conclude $A \; true$ if we can verify all assumptions in Ψ.

$$\frac{\Delta, A \; valid \; \Psi, \Delta'; \Gamma \vdash \Psi}{\Delta, A \; valid \; \Psi, \Delta'; \Gamma \vdash A \; true} \; (*)$$

In other words, if we know $A \; true$ in Ψ, and all elements in Ψ can be verified from the assumptions in Γ, then we can conclude $A \; true$ in Γ. As we will see in the next section, this transition from one context Ψ to another context Γ, can be achieved via a substitutions from Ψ to Γ.

Dual to the above development of relative validity C *valid* Ψ, we can introduce simultaneous possibility (Ψ, C) *poss*. In terms of multiple world semantics, this means that there exists a world in which C_1 *true*, ..., C_n *true*, C *true*. The possibility judgment now has the more complex form

$$A_1 \textit{ valid } \Psi_1, \ldots, A_n \textit{ valid } \Psi_n; B_1 \textit{ true}, \ldots, B_m \textit{ true} \vdash (\Psi, C) \textit{ poss}.$$

If a simultaneous possibility (Ψ, A) *poss* is established, it can be used to derive new possibilities.

$$\frac{\Delta, \Gamma \vdash (\Psi, A) \textit{ poss} \quad \Delta; (\Psi, A) \vdash (\Psi', C) \textit{ poss}}{\Delta; \Gamma \vdash (\Psi', C) \textit{ poss}} \;(**)$$

While it is interesting to investigate this modal logic above in its own right, we will focus on corresponding proof term calculus.

2.2 Dependently typed calculus with modal variables

In this section, we introduce a dependently typed lambda calculus. We distinguish between modal variables $u{::}(\Psi{\vdash}A)$ for relative validity assumptions A *valid* Ψ declared in a modal context, and $x{:}A$ for ordinary truth assumptions A *true* declared in an (ordinary) context. Note that the substitution σ is part of the syntax of modal variables. c and a are constants, which are declared in a signature. This is a conservative extension of the LF [6] so we suppress some routine details such as signatures.

$$\begin{array}{rrcl}
\text{Kinds} & K & ::= & \mathsf{type} \mid \Pi x{:}A.\, K \\
\text{Families} & A, B, C & ::= & a \mid A\, M \mid \Pi x{:}A_1.\, A_2 \\
\text{Objects} & M, N & ::= & c \mid x \mid u[\sigma] \mid \lambda x{:}A.\, M \mid M_1\, M_2 \\
\text{Substitutions} & \sigma, \tau & ::= & \cdot \mid \sigma, M/x \\
\text{Contexts} & \Gamma, \Psi & ::= & \cdot \mid \Gamma, x{:}A \\
\text{Modal Contexts} & \Delta & ::= & \cdot \mid \Delta, u{::}(\Psi{\vdash}A)
\end{array}$$

The principal judgments are listed below. As usual, we omit similar judgments on types and kinds and all judgments concerning definitional equality.

$\Delta; \Gamma \vdash M : A$	Object M has type A
$\Delta; \Gamma \vdash \sigma : \Psi$	Substitution σ matches context Ψ
$\vdash \Delta\ \mathsf{mctx}$	Δ is a valid modal context
$\Delta \vdash \Psi\ \mathsf{ctx}$	Ψ is a valid context

We will tacitly rename bound variables, and maintain that contexts and substitutions declare no variable more than once. Note that substitutions σ are defined only on ordinary variables x and not modal variables u. We write id_Γ for the identity substitution $(x_1/x_1, \ldots, x_n/x_n)$ for a context $\Gamma = (\cdot, x_1{:}A_1, \ldots, x_n{:}A_n)$. We also streamline the calculus slightly by always substituting simultaneously for all ordinary variables. This is not essential, but saves some tedium in relating simultaneous and iterated substitution. The typing rules are given below.

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

$$\frac{\Delta; \Gamma, x{:}A_1 \vdash M : A_2}{\Delta; \Gamma \vdash \lambda x{:}A_1.\, M : \Pi x{:}A_1.\, A_2} \qquad \frac{\Delta; \Gamma \vdash M_1 : \Pi x{:}A_2.\, A_1 \quad \Delta; \Gamma \vdash M_2 : A_2}{\Delta; \Gamma \vdash M_1\, M_2 : [\mathsf{id}_\Gamma, M_2/x]A_1}$$

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

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

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

Note that the rule for modal variables is the rule (*) presented in the previous section, annotated with proof terms and slightly generalized, because of the dependent type theory we are working in.

Our convention is that substitutions as defined operations on expressions are written in prefix notation $[\sigma]P$ for an object, family, kind, or substitution P. These operations are capture-avoiding as usual. Moreover, we always assume that all free variables in P are declared in σ. Substitutions that are part of the syntax are written in postfix notation, $u[\sigma]$. Note that such explicit substitutions occur only for variables u labeling relative validity assumptions.

Substitutions are defined in a standard manner. We omit the details at the level of types and kinds for the sake of brevity.

$$\begin{array}{rcll}
[\sigma]c & = & c & \\
[\sigma_1, M/x, \sigma_2]x & = & M & \\
[\sigma](u[\tau]) & = & u[[\sigma]\tau] & \\
[\sigma](N_1\, N_2) & = & ([\sigma]N_1)\,([\sigma]N_2) & \\
[\sigma](\lambda y{:}A.\ N) & = & \lambda y{:}[\sigma]A.\ [\sigma, y/y]N & \text{provided } y \text{ not declared or free in } \sigma \\
 & & & \\
[\sigma](\cdot) & = & (\cdot) & \\
[\sigma](\tau, N/y) & = & ([\sigma]\tau, [\sigma]N/y) & \text{provided } y \text{ not declared or free in } \sigma
\end{array}$$

The side conditions can always be verified by (tacitly) renaming bound variables. We do not need an operation of applying a substitution σ to a context. The last principle makes it clear that $[\sigma]\tau$ corresponds to composition of substitutions, which is sometimes written as $\tau \circ \sigma$.

The following substitution principles for substitutions σ hold. They are suggested by the modal interpretation and proved by simple structural inductions. We elide corresponding principles for families and kinds.

Theorem 1 (Explicit substitutions)

1. *If* $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Psi \vdash N : C$ *then* $\Delta;\Gamma \vdash [\sigma]N : [\sigma]C$.
2. *If* $\Delta;\Gamma \vdash \sigma : \Psi$ *and* $\Delta;\Psi \vdash \tau : \Psi'$ *then* $\Delta;\Gamma \vdash [\sigma]\tau : \Psi'$.
3. $[\sigma]([\tau]M) = [[\sigma]\tau]M$ *and* $[\sigma]([\tau]\tau') = [[\sigma]\tau]\tau'$

A new and interesting operation realizes the substitution principles for relative validity in Theorem 2 below. The new operation is denoted as $[\![M/u]\!]$. It substitutes an object M for a *modal* variable u, and we define it below. The substitution is compositional, but two interesting situations arise: when a variable u is encountered, and when we substitute into a λ-abstraction. For sake of brevity, we only give the substitution on objects.

$$\begin{array}{rcl}
[\![M/u]\!]c & = & c \\
[\![M/u]\!]x & = & x \\
[\![M/u]\!](u[\sigma]) & = & [[\![M/u]\!]\sigma]M \\
[\![M/u]\!](v[\sigma]) & = & v[[\![M/u]\!]\sigma] \quad \text{for } u \neq v \\
[\![M/u]\!](N_1\, N_2) & = & ([\![M/u]\!]N_1)\,([\![M/u]\!]N_2) \\
[\![M/u]\!](\lambda y{:}A.\ N) & = & \lambda y{:}[\![M/u]\!]A.\ [\![M/u]\!]N
\end{array}$$

We remark that the rule for substitution into a λ-abstraction does not require a side condition. This is because the object M is defined in a different context, which is accounted for by the explicit substitution stored at occurrences of u.

Finally, consider the case of substituting into a closure, which is the critical case of this definition.

$$[\![M/u]\!](u[\sigma]) \quad = \quad [[\![M/u]\!]\sigma]M$$

This is clearly well-founded, because σ is a subexpression (so $[\![M/u]\!]\sigma$ will terminate) and application of an ordinary substitution has been defined previously without reference to the new form of substitution.

Similar to the substitution properties for ordinary explicit substitutions, we now can show that the new substitution operation for relative validity satisfies the substitution principles. This highlights the interplay between ordinary and modal substitutions. Again, this is motivated by the logical interpretation and follows by simple inductions after straightforward generalization to encompass all syntactic categories.

Theorem 2 (Modal substitutions)

1. *If* $\Delta; \Psi \vdash M : A$ *and* $\Delta, u{::}(\Psi{\vdash}A), \Delta'; \Gamma \vdash N : C$ *then* $\Delta, [\![M/u]\!]\Delta'; [\![M/u]\!]\Gamma \vdash [\![M/u]\!]N : [\![M/u]\!]C$
2. *If* $\Delta; \Psi \vdash M : A$ *and* $\Delta, u{::}(\Psi{\vdash}A), \Delta'; \Gamma \vdash \tau : \Psi'$ *then* $\Delta, [\![M/u]\!]\Delta'; [\![M/u]\!]\Gamma \vdash [\![M/u]\!]\tau : [\![M/u]\!]\Psi'$
3. $[\![M/u]\!]([\sigma]P) = [[\![M/u]\!]\sigma]([\![M/u]\!]P)$
4. $[\![M/u]\!]([\![N/v]\!]P) = [\![[\![M/u]\!]N/v]\!]([\![M/u]\!]P)$ *if* $u \neq v$ *and* v *not free in* M

3 Modal necessity

The judgmental reconstruction in terms of modal logic suggests that there should be two simple and clean ways to incorporate such variables. One is via a new quantifier $\Pi^{\square}u{::}(\Psi{\vdash}A_1).\ A_2$, the other is via a general modal operator $\square_{\Psi}$. Proof-theoretically, the former is simpler, so we will pursue this here.

$$\begin{array}{lrcl} \text{Families} & A, B, C & ::= & \ldots \mid \Pi^{\square}u{::}A_1.\ A_2 \\ \text{Objects} & M, N & ::= & \ldots \mid \lambda^{\square}u{::}A.M \mid M_1 \mathbin{\square} M_2 \end{array}$$

Next, we give the typing rules. Next, we give introduction and elimination forms for the new objects.

$$\frac{\Delta; \Psi \vdash A : \mathsf{type} \quad \Delta, u{::}(\Psi{\vdash}A); \Gamma \vdash B : \mathsf{type}}{\Delta; \Gamma \vdash \Pi^{\square}u{::}(\Psi{\vdash}A).\ B : \mathsf{type}}$$

$$\frac{\Delta, u{::}(\Psi{\vdash}A); \Gamma \vdash M : B}{\Delta; \Gamma \vdash \lambda^{\square}u{::}(\Psi{\vdash}A).\ M : \Pi^{\square}u{::}(\Psi{\vdash}A).\ B} \qquad \frac{\Delta; \Gamma \vdash N : \Pi^{\square}u{::}(\Psi{\vdash}A).\ B \quad \Delta; \Psi \vdash M : A}{\Delta; \Gamma \vdash N \mathbin{\square} M : [\![M/u]\!]B}$$

The main complication of this extension is that variables u can now bound and substitution must be capture avoiding. We note that the context Ψ and the type A in the type family $\Pi^{\square}u{::}(\Psi{\vdash}A)$ are separate from the context Γ. In other words elements in the context Ψ may depend on each other, but they may not depend on Γ. Similarly, the type family A, only depends on the context Ψ and not on the context Γ. In accordance with this comment, explicit substitutions need only descend into the body of the $\lambda^{\square}$. Similarly, in a box-application $N \mathbin{\square} M$ we do not need to apply the substitution σ to M, since M is well-typed in a different context. Operation of explicit substitution is extended with the new cases as follows.

$$\begin{array}{rcl} [\sigma](N_1 \mathbin{\square} N_2) & = & ([\sigma]N_1) \mathbin{\square} N_2 \\ [\sigma](\lambda^{\square}u{::}(\Psi{\vdash}A).N) & = & \lambda^{\square}u{::}(\Psi{\vdash}A)[\sigma]N \end{array}$$

Next, we extend the modal substitutions. This is straightforward.

$$\begin{array}{rcl} [\![M/u]\!](\lambda^{\square}v{::}(\Psi{\vdash}A).N) & = & \lambda^{\square}v{::}([\![M/u]\!]\Psi{\vdash}[\![M/u]\!]A).[\![M/u]\!]N \quad \text{for } u \neq v \\ [\![M/u]\!](N_1 \mathbin{\square} N_2) & = & ([\![M/u]\!]N_1) \mathbin{\square} ([\![M/u]\!]N_2) \end{array}$$

The proofs of the substitution principles stated in Theorem 1 and Theorem 2 also readily extend. The extension does not require any changes to the statement of these theorems.

4 Modal possibility

This section introduces a proof-term assignment for the judgment of simultaneous possibility and internalizes this judgment via a new modal family $\Diamond_\Psi$. We summarize below the changes to the syntax of the calculus.

$$\begin{array}{lrcl} \text{Families} & A & ::= & \ldots \mid \Diamond_\Psi A \\ \text{Frames} & E, F & ::= & \langle \sigma, M \rangle \mid \textbf{let dia } x = M \textbf{ in } E \\ \text{Objects} & M, N & ::= & \ldots \mid \textbf{dia } E \end{array}$$

As families now contain a new modal constructor, the kinding judgment has to be extended to account for this new case.

$$\frac{\Delta; \Psi \vdash A : \mathsf{type}}{\Delta; \Gamma \vdash \Diamond_\Psi A : \mathsf{type}}$$

The above rule allows the type A to depend on the context Ψ, but it prevents both Ψ and A to depend on Γ. This property will facilitate the proof-term assignment for the judgment of simultaneous possibility. To that end, the elements of the syntactic category of frames will serve as appropriate proof-terms. The judgment of simultaneous possibility now has the following syntactic form:

$$\Delta; \Gamma \vdash E \div \langle \Psi \rangle A.$$

The judgment establishes that Ψ and A are simultaneously possible, and the frame E is a witness for that. Notice that A may depend on Ψ, and E may depend on Γ. It is assumed, however, that neither A nor Ψ may depend on Γ. As will be shown later, this is a crucial invariant, and we will maintain it in all the rules of the judgment.

In order to internalize simultaneous possibility, we use a new object constructor **dia** to witness the following correspondence: if Ψ and A are *simultaneously possible*, then $\Diamond_\Psi A$ is *true*. When decorated with frames and objects that witness the appropriate truths and possibilities, this correspondence can be phrased as the typing rule below. Because of the assumed independence of Ψ and A from Γ, the family $\Diamond_\Psi A$ will be well-formed.

$$\frac{\Delta; \Gamma \vdash E \div \langle \Psi \rangle A}{\Delta; \Gamma \vdash \textbf{dia } E : \Diamond_\Psi A}$$

The rules for simultaneous possibility are derived from the definitional properties of possibility presented in Section 2.1.

$$\frac{\Delta; \Gamma \vdash \sigma : \Psi \quad \Delta; \Gamma \vdash M : [\sigma]A}{\Delta; \Gamma \vdash \langle \sigma, M \rangle \div \langle \Psi \rangle A} \qquad \frac{\Delta; \Gamma \vdash M : \Diamond_\Psi A \quad \Delta; (\Psi, x{:}A) \vdash E \div \langle \Psi' \rangle B}{\Delta; \Gamma \vdash \textbf{let dia } x = M \textbf{ in } E \div \langle \Psi' \rangle B} \,(**)$$

The first rule corresponds to the definitional property (1): if Ψ and A are simultaneously true, then they certainly are simultaneously possible. As can be noticed, a witness for a simultaneous possibility of Ψ and A is a pair consisting of a substitution σ witnessing the truth of Ψ, and an object M witnessing the truth of $[\sigma]A$. This justifies the comparison of simultaneous possibility with dependent sums; where dependent sums quantify over a single variable, the simultaneous possibility quantifies over an arbitrary context Ψ.

The rule (**) parallels the definitional property (2), and the typing rule (**) from Section 2.1, with a small modification. Here we use the internalized form $\Diamond_\Psi A$ of the simultaneous possibility of Ψ and A, so that the rule also serves to eliminate the modality. As explained before, it is a maintained invariant that Ψ' and B in the second premise are independent of $\Psi, x{:}A$. As a consequence, there are no dependencies in $\langle \Psi' \rangle B$ arising in the second premise, that need to be removed from the conclusion of the rule. This is why the mentioned invariant is crucial for the formulation of the judgment.

The definitions of explicit and modal substitutions readily extend to frames, as do the corresponding substitutions principles. The substitution principles on objects are also easily updated to account for the new object constructor **dia**.

$$\begin{array}{rcl}
[\sigma](\Diamond_\Psi A) & = & \Diamond_\Psi A \\
[\sigma](\mathbf{dia}\ E) & = & \mathbf{dia}\ [\sigma]E \\
[\sigma]\langle\tau, M\rangle & = & \langle[\sigma]\tau, [\sigma]M\rangle \\
[\sigma](\mathbf{let\ dia}\ x = M\ \mathbf{in}\ E) & = & \mathbf{let\ dia}\ x = [\sigma]M\ \mathbf{in}\ E
\end{array}$$

Notice that in the last clause of the above definition, the substitution σ is not applied to the body E of **let dia**. Indeed, E is defined in a context provided by M, and hence does not share any variables with σ. This property is also reflected in the following theorem which formulates the explicit substitution principle for frames. Observe that the types in the concluding judgment of the principle are not modified by the substitution σ, which is in contrast to the corresponding principles for objects.

Theorem 3 (Explicit substitutions for frames)

1. *If* $\Delta; \Gamma \vdash \sigma : \Psi$, *and* $\Delta; \Psi \vdash E \div \langle\Psi'\rangle A$, *then* $\Delta; \Gamma \vdash [\sigma]E \div \langle\Psi'\rangle A$.
2. $[\sigma]([\tau]E) = [[\sigma]\tau]E$

The operation of modal substitution commutes with all the new constructors.

$$\begin{array}{rcl}
[\![M/u]\!](\Diamond_\Psi A) & = & \Diamond_{([\![M/u]\!]\Psi)}[\![M/u]\!]A \\
[\![M/u]\!](\mathbf{dia}\ E) & = & \mathbf{dia}\ [\![M/u]\!]E \\
[\![M/u]\!]\langle\tau, N\rangle & = & \langle[\![M/u]\!]\tau, [\![M/u]\!]N\rangle \\
[\![M/u]\!](\mathbf{let\ dia}\ x = N\ \mathbf{in}\ E) & = & \mathbf{let\ dia}\ x = [\![M/u]\!]N\ \mathbf{in}\ [\![M/u]\!]E \quad \text{if } x \text{ not free in } M
\end{array}$$

Theorem 4 (Modal substitutions for frames)
If $\Delta; \Psi \vdash M : A$ *and* $\Delta, u{::}(\Psi \vdash A), \Delta'; \Gamma \vdash E \div \langle\Psi'\rangle B$, *then*
$\Delta, [\![M/u]\!]\Delta'; [\![M/u]\!]\Gamma \vdash [\![M/u]\!]E \div \langle[\![M/u]\!]\Psi'\rangle[\![M/u]\!]B$.

The most interesting development, however, is the formulation of the new operation of frame substitution. It is justified by the definitional property (2) from Section 2.1, and its obvious generalization to simultaneous possibility which we have phrased as the typing rule (**). Frame substitution is therefore an operation transforming the frame $F \div \langle\Psi\rangle A$ and the frame E such that $\Psi, x{:}A \vdash E \div \langle\Psi'\rangle B$ into a frame $\langle\!\langle F/x\rangle\!\rangle E$ which is supposed to witness the simultaneous possibility $\langle\Psi'\rangle B$. Curiously, the operation is defined by induction on the structure of F rather than E:

$$\begin{array}{rcl}
\langle\!\langle\langle\sigma, M\rangle/x\rangle\!\rangle E & = & [\sigma, M/x]E \\
\langle\!\langle\mathbf{let\ dia}\ y = M\ \mathbf{in}\ F/x\rangle\!\rangle E & = & \mathbf{let\ dia}\ y = M\ \mathbf{in}\ \langle\!\langle F/x\rangle\!\rangle E
\end{array}$$

Lastly, we need a theorem to establish the soundness of the operation, i.e. the fact that the resulting frame $\langle\!\langle F/x\rangle\!\rangle E$ indeed is a witness for the required simultaneous possibility.

Theorem 5 (Frame substitution principle)
If $\Delta; \Gamma \vdash F \div \langle\Psi\rangle A$, *and* $\Delta; (\Psi, x{:}A) \vdash E \div \langle\Psi'\rangle B$, *then* $\Delta; \Gamma \vdash \langle\!\langle F/x\rangle\!\rangle E \div \langle\Psi'\rangle B$.

5 Conclusions and future work

In this paper we have presented a dependently typed modal calculus admitting the notions of relative validity and simultaneous possibility. Both notions are dependent on a context Ψ, and have been beneficial in several related systems that we have considered before [14, 8, 9]. In the present calculus, we internalize the simultaneous possibility by introducing a family of modal operators $\Diamond_\Psi$. The operators are indexed by dependently typed (non-modal) contexts Ψ, and in $\Diamond_\Psi A$, the type A is allowed to depend on variables in Ψ.

While we allow universal quantification over valid variables, we did not internalize relative validity in the same way. Non-relative validity of propositional Constructive S4 is usually formulated by means of proof terms **box** and **let box** with the following typing rules

$$\frac{\Delta; \cdot \vdash M : A}{\Delta; \Gamma \vdash \mathbf{box}\ M : \Box A} \qquad \frac{\Delta; \Gamma \vdash M : \Box A \quad \Delta, u{::}A; \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = M\ \mathbf{in}\ N : B}$$

Naive extension to relative validity may be attempted by the following reformulation

$$\frac{\Delta; \Psi \vdash M : A}{\Delta; \Gamma \vdash \mathbf{box}\ M : \Box_\Psi A} \qquad \frac{\Delta; \Gamma \vdash M : \Box_\Psi A \quad \Delta, u{::}(\Psi{\vdash}A); \Gamma \vdash N : B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = M\ \mathbf{in}\ N : B}$$

However, this would not be a sound extension. Indeed in the above **let box** rule, the type B in the second premise may depend on the modal variable u, but there is no simple way to eliminate this dependency in the conclusion. Thus, as immediate future work, we plan to explore the approach taken in [2], where the **let box** is eliminated in favor of a simpler **unbox**. The price to pay is that instead of maintaining only two variable contexts Δ and Γ, we would have to maintain a whole stack of contexts $\Gamma_1; \Gamma_2; \cdots; \Gamma_n$. Similar formulation of the necessitation modality (albeit of a non-indexed version), has already been employed in a dependent setup in [3].

Another important missing element is the addition of equality judgment on objects and families, which we need in order to formulate a full-fledged modal type theory. The core dependent calculus with no modal operators, but with modal variables and universal quantification over modal variables, certainly admits well-behaved notions of equality and canonical forms [13]. We conjecture that the fragment extended with possibility that we presented here also admits canonical forms. The only difficulty may arise from the presence of the **let dia** form. Let constructs typically invalidate the canonical forms lemma of logical frameworks because they introduce commuting conversions. But **let dia** may actually prove benign as it witnesses a judgment of monadic form [15]. However, these questions, as well as the equational properties of the extension with the family $\Box_\Psi$ of modal operators, remain for future work.

Finally, some applications that we have in mind for the new calculus concern the use of modalities in the representation of proofs and propositions. For example, the modality $\Diamond$ may be used to combine a set of notational definitions with a proposition expressed in terms of these definitions. The necessitation modality $\Box$ and its corresponding proof objects could be used to represent proofs in a compact form where common subexpressions are explicitly named and reused.

References

[1] Natasha Alechina, Michael Mendler, Valeria de Paiva, and Eike Ritter. Categorical and Kripke semantics for Constructive S4 modal logic. In Laurent Fribourg, editor, *International Workshop on Computer Science Logic, CSL'01*, volume 2142 of *Lecture Notes in Computer Science*, pages 292–307, Paris, France, 2001. Springer.

[2] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[3] Joëlle Despeyroux and Pierre Leleu. Primitive recursion for higher-order abstract syntax with dependent types. In *International Workshop on Intuitionistic Modal Logics and Applications (IMLA)*, pages 555–572, 1999.

[4] Joëlle Despeyroux and Pierre Leleu. Recursion over objects of functional type. *Mathematical Structures in Computer Science*, 11(4):555–572, 2001.

[5] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, April 1997. Springer-Verlag LNCS. An extended version is available as Technical Report CMU-CS-96-172, Carnegie Mellon University.

[6] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[7] Peter Lee and Mark Leone. Optimizing ML with run-time code generation. In *Conference on Programming Language Design and Implementation, PLDI'96*, pages 137–148, 1996.

[8] Aleksandar Nanevski. Meta-programming with names and necessity. In *International Conference on Functional Programming, ICFP'02*, pages 206–217, Pittsburgh, Pennsylvania, 2002. A significant revision is available as a technical report CMU-CS-02-123R, Computer Science Department, Carnegie Mellon University.

[9] Aleksandar Nanevski. From dynamic binding to state via modal possibility. In *International Conference on Principles and Practice of Declarative Programming, PPDP'03*, Uppsala, Sweden, 2003. To appear.

[10] Michael Florentin Nielsen. Combining closed and open code. Unpublished, 2001.

[11] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[12] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[13] Brigitte Pientka. *Tabled higher-order logic programming.* PhD thesis, Department of Computer Sciences, Carnegie Mellon University, 2003. forthcoming.

[14] Brigitte Pientka and Frank Pfennning. Optimizing higher-order pattern unification. In F. Baader, editor, *19th International Conference on Automated Deduction, Miami, USA*, Lecture Notes in Computer Science (LNCS), to appear. Springer, 2003.

[15] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002.

[16] Philip Wickline, Peter Lee, and Frank Pfenning. Run-time code generation and Modal-ML. In *Conference on Programming Language Design and Implementation, PLDI'98*, pages 224–235, Montreal, Canada, 1998.

[17] Philip Wickline, Peter Lee, Frank Pfenning, and Rowan Davies. Modal types as staging specifications for run-time code generation. *ACM Computing Surveys*, 30(3es), 1998.