Electronic Notes in Theoretical Computer Science 1 (1995)
URL: http://www.elsevier.nl/locate/entcs/volume1.html 20 pages

On a Modal λ-Calculus for S4*

F. Pfenning

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.
fp@cs.cmu.edu

H. C. Wong

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.
hcwong@cs.cmu.edu

Abstract

We present $\lambda^{\rightarrow\Box}$, a concise formulation of a proof term calculus for the intuitionistic modal logic S4 that is well-suited for practical applications. We show that, with respect to provability, it is equivalent to other formulations in the literature, sketch a simple type checking algorithm, and prove subject reduction and the existence of canonical forms for well-typed terms. Applications include a new formulation of natural deduction for intuitionistic linear logic, modal logical frameworks, and a logical analysis of staged computation and binding-time analysis for functional languages [6].

1 Introduction

Modal operators familiar from traditional logic have received renewed attention in computer science through their importance in linear logic. Typically, they are described axiomatically in the style of Hilbert or via sequent calculi. However, the Curry-Howard isomorphism between proofs and λ-terms is most poignant for natural deduction, so natural deduction formulations of modal and linear logics have also been the subject of research [20,1,5,3,4,22,13]. Although most of the researchers in this area agree on the (potential) applications of modal logics, the works published to date seem to be primarily oriented toward proof theory and not well suited for practical applications.

* This work is supported by NSF Grant CCR-9303383 and the Advanced Research Projects Agency under ARPA Order No. 8313.

Among the application areas of modal and linear logics are those of functional programming and logical frameworks. In these settings, the simplicity of the language, a feasible type checking procedure, the existence of canonical forms, and the property of type preservation are among the most desirable results. None of the work done to date, however, incorporates these results and characteristics.

In this paper we present $\lambda^{\rightarrow\Box}$, a new, concise formulation of a term calculus equivalent, via a Curry-Howard isomorphism, to the intuitionistic modal logic S4, that is well-suited for type checking. We show that, with respect to provability, it is equivalent to other formulations in the literature and sketch a type checking algorithm. We then prove subject reduction and the existence of a canonical form for every well-typed term. We omit a simpler and not so relevant strong normalization result. A similar, but more verbose system motivated by proof-theoretic considerations has been given by Martini & Masini [13]. They investigate normalization and the Church-Rosser property which is complementary to our own meta-theoretic study.

A similar approach can be used beyond S4 to give an analogous formulation of intuitionistic linear logic which is beyond the scope of this paper. A related system without weakening and contraction has been analysed by Martini & Masini [14]. Our system can also serve as the basis for a modal logical framework used for explanation-based generalization [7] and for binding-time analysis of functional languages [6]. Another application to the combination of higher-order abstract syntax and induction in logical frameworks is the subject of current research and sketched in slightly more detail in the conclusion.

The remainder of this paper is structured as follows. Section 2 presents what we call an *explicit system*, as given in [5,22], a formulation of a modal λ-calculus for S4 in which explicit substitutions in terms are used to enforce the global validity conditions associated with the modal $\Box$ operator. In Section 3 we present our system $\lambda^{\rightarrow\Box}$, which immediately yields a practical procedure for type checking. In Section 4 we show that the two are equivalent with respect to provability. In Section 5 we introduce a notion of approximate typing and show that every approximately typed term can be converted to canonical form. We also prove subject reduction and type preservation for precise typing. In Section 6, we discuss related work. In Section 7 we conclude with future work and brief discussions about concrete and potential applications.

2 Modal λ-Calculus: An Explicit Formulation

In this section we present a formulation of a modal λ-calculus for intuitionistic S4, as it is given, for example, in [5]. We denote this system by $\lambda_e^{\rightarrow\Box}$. We refer to it as *explicit* since it requires substitutions to occur explicitly within terms in order to deal with the non-local validity conditions imposed by the modal operator. Much of the material in the subsequent sections will be concerned with a formulation where substitutions are no longer required.

2.1 Syntax

Types	$A ::= b \mid A_1 \to A_2 \mid \Box A$
Terms	$E ::= x \mid \lambda x{:}A.\ E \mid E_1\ E_2 \mid \mathrm{box}^{\sigma}_{\Box\Gamma} E \mid \mathrm{unbox}\ E$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x{:}A$
Substitutions	$\sigma ::= \cdot \mid \sigma, E/x$
Boxed Contexts	$\Box\Gamma ::= \cdot \mid \Box\Gamma, x{:}\Box A$

We use b for base types and x for variables. We assume that any variable can be declared at most once in a context Γ and defined at most once in a substutition σ. Bound variables may be renamed tacitly. We use A, B to range over types, E to range over terms, and Γ, Δ to range over contexts. We omit leading $\cdot$'s from contexts and substitutions for the sake of brevity. We write $[E'/x]E$ for the result of substituting E' for x in E, renaming bound variables as necessary in order to avoid the capture of free variables in E'. Similarly, σE is the result of applying the substitution σ to E. We write id_Γ for the identity substution $x_1/x_1, \ldots, x_n/x_n$ on $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$. The addition of types $\Box A$ to the simply-typed λ-calculus introduces two new term constructs: *box* and *unbox*. *box* introduces objects of type $\Box A$, and *unbox* is the corresponding elimination construct. The *box* construct requires a substitution σ and a context $\Box\Gamma$ whose role should become clear when we present the typing rules.

2.2 Typing Rules

The problem of typing in the explicit system is well understood; we present here the system in [5], using a slightly different notation. It is given by the following set of inference rules defining the mutually recursive judgments for the typing of terms and substitutions.

$\Delta \vdash^e E : A$ explicit term E has type A in context Δ

$\Delta \vdash^e \sigma : \Gamma$ σ is a valid substitution from terms over Γ to terms over Δ

$$\frac{x{:}A \text{ in } \Delta}{\Delta \vdash^e x : A}\ var$$

$$\frac{\Delta, x{:}A \vdash^e E : B}{\Delta \vdash^e \lambda x{:}A.\ E : A \to B} \to I \qquad \frac{\Delta \vdash^e E_1 : B \to A \qquad \Delta \vdash^e E_2 : B}{\Delta \vdash^e E_1\ E_2 : A} \to E$$

$$\frac{\Delta \vdash^e \sigma : \Box\Gamma \qquad \Box\Gamma \vdash^e E : A}{\Delta \vdash^e \mathrm{box}^{\sigma}_{\Box\Gamma} E : \Box A} \Box I \qquad \frac{\Delta \vdash^e E : \Box A}{\Delta \vdash^e \mathrm{unbox}\ E : A} \Box E$$

$$\frac{}{\Delta \vdash^e \cdot : \cdot}\ sub1 \qquad \frac{\Delta \vdash^e \sigma : \Gamma \qquad \Delta \vdash^e E : A}{\Delta \vdash^e (\sigma, E/x) : (\Gamma, x{:}A)}\ sub2$$

We do not show some elementary properties of substitutions, such as $\Gamma \vdash^e id_\Gamma : \Gamma$. At this point the importance of σ and $\Box\Gamma$ in a term $\mathrm{box}^{\sigma}_{\Box\Gamma} E$ should be apparent: The substitution σ must provide well-typed terms for all variables

in $\Box\Gamma$ and E must be well typed in $\Box\Gamma$ alone. It is this restriction which enforces the correct use of the $\Box$ operator in accordance with the laws of S4. The interested reader is referred to [5] for a further discussion and relationship to sequent and other formulations. Before we proceed to the next section, however, we would like to point out that an apparently simpler version of $\Box$-introduction,

$$\frac{\Box\Gamma \vdash M : A}{\Box\Gamma \vdash \mathrm{box}\, M : \Box A}$$

is inadequate since subject reduction fails (a fact noted by Troelstra [22]). Consider the system obtained from $\lambda_e^{\to\Box}$ by replacing its introduction rule for $\Box$ by the just-mentioned rule. A concrete example that demonstrates the failure of subject reduction is the term $(\lambda z{:}\Box A.\ \mathrm{box}\, z)\,(y\, x)$ under the context $x{:}B, y{:}B \to \Box A$. It is easy to check that

$$x{:}B, y{:}B \to \Box A \vdash (\lambda z{:}\Box A.\ \mathrm{box}\, z)\,(y\, x) : \Box\Box A.$$

However, after performing a β-reduction we obtain $\mathrm{box}\,(y\, x)$ and

$$x{:}B, y{:}B \to \Box A \not\vdash \mathrm{box}\,(y\, x) : \Box\Box A.$$

The subject reduction property is, therefore, violated.

In $\lambda_e^{\to\Box}$, $(\lambda z{:}\Box A.\ \mathrm{box}\, z)\,(y\, x)$ might be written as $(\lambda z{:}\Box A.\ \mathrm{box}^{z/z}_{z:\Box A}\, z)\,(y\, x)$ where

$$x{:}B, y{:}B \to \Box A \vdash^{e} (\lambda z{:}\Box A.\ \mathrm{box}^{z/z}_{z:\Box A}\, z)\,(y\, x) : \Box\Box A.$$

The result of a β-reduction would be $\mathrm{box}^{(yx)/z}_{z:\Box A} z$ and it is easy to check that

$$x{:}B, y{:}B \to \Box A \vdash^{e} \mathrm{box}^{(yx)/z}_{z:\Box A} z : \Box\Box A.$$

$\lambda_e^{\to\Box}$ has a number of well-known pleasant properties, such as uniqueness of the typing derivation given a well-typed term and a context, and the subject reduction property assuming a standard notion of reduction following [4]. On the other hand, the calculus lacks a clear notion of normal form, since substitutions may interfere with redices. We elaborate on this at the beginning of the next section. Furthermore, its equational theory requires a large number of unpleasant commutative conversions.

3 Modal λ-Calculus: An Implicit Formulation

The explicit formulation for $\lambda_e^{\to\Box}$, as presented in the previous section, has two disadvantages: it is extremely cumbersome to write down non-trivial terms and it is hard to detect the presence of redices. The latter problem is exemplified by the term

$$\mathrm{box}^{(\mathrm{box}\,(\lambda x{:}b.\ x))/f}_{f:\Box(b\to b)} \mathrm{unbox}\, f\, c,$$

assuming we have a base type b and a constant $c{:}b$. The *box* operator hides redices; here it is the application of unbox to f (which stands for $\mathrm{box}(\lambda x{:}b.\ x)$) and, if that is reduced, the application of $(\lambda x{:}b.\ x)$ to c.

This observation is not new and one solution has been proposed by Troelstra [22] in the context of linear logic which we view as a modern reconstruction

of Prawitz's natural deduction for S4 [20]. In Troelstra's system the *box* operator would no longer contain a substitution and the rule $\Box I$ would be replaced by

$$\frac{\Delta \vdash^e \sigma : \Box\Gamma \qquad \Delta \vdash^e \sigma E : A}{\Delta \vdash^e \mathrm{box}\,(\sigma E) : \Box A}\,\Box I^+$$

with the side condition that the free variables of E are contained in $\Box\Gamma$. Note that σE in the conclusion and premise of this rule is the result of applying σ to E; σ must therefore be guessed during the type-checking process. This avoids the question of how to actually perform type checking—interpreting the rules literally is clearly infeasible. This system is intermediate in the sense that substitutions no longer occur explicitly in terms, but they do occur in typing derivations. Eliminating them altogether is the subject of the remainder of this section. Terms like $\mathrm{box}^{\sigma}_{\Box\Gamma}\, E$ will also be represented as $\mathrm{box}\, M$, where M corresponds to σE, but the conditions on valid introductions of $\Box$ are enforced in a different way which directly leads to a type checking procedure. The central idea is that of a *context stack* introduced below.

3.1 Syntax

Types	$A ::= b \mid A_1 \to A_2 \mid \Box A$
Terms	$M ::= x \mid \lambda x{:}A.\, M \mid M_1\, M_2 \mid \mathrm{box}\, M \mid \mathrm{unbox}\, M$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x{:}A$
Context Stacks	$\Psi ::= \cdot \mid \Psi;\Gamma$

All the categories, except *context stacks* are standard. The importance of context stacks will be apparent when we present the typing rules. They implement a possible world semantics in a syntactic way, whereby each context represents a world from which we can access all the worlds to its right (including itself). Consequently the order of the contexts inside a stack is important, while the ordering of variables within each context is irrelevant. Notice that *box* terms no longer have explicit substitutions associated with them. Also, a variable may occur at most once in an entire stack (not just in a context).

3.2 Typing Rules

In this section we present typing rules for the implicit formulation of $\lambda^{\to\Box}$ using context stacks. They enable the distinguished use of variables depending on the relative position of their declarations with respect to the *box* operators that enclose the term being typed. We present the typing judgment along with some examples showing the interesting steps of the typing process. This should provide some intuition about the way our typing rules enforce modal restrictions and how a type checking procedure would operate.

Formally, the typing judgement

$\Psi; \Gamma \stackrel{i}{\vdash} M : A$ implicit term M has type A in local context Γ under stack Ψ

is defined by the following inference rules:

$$\frac{x{:}A \text{ in } \Gamma}{\Psi; \Gamma \stackrel{i}{\vdash} x : A}\, var \qquad \frac{\Psi; (\Gamma, x{:}A) \stackrel{i}{\vdash} M : B}{\Psi; \Gamma \stackrel{i}{\vdash} \lambda x{:}A.\, M : A \to B} \to I$$

$$\frac{\Psi; \Gamma \stackrel{i}{\vdash} M : B \to A \qquad \Psi; \Gamma \stackrel{i}{\vdash} N : B}{\Psi; \Gamma \stackrel{i}{\vdash} M\, N : A} \to E$$

$$\frac{\Psi; \Gamma; \cdot \stackrel{i}{\vdash} M : A}{\Psi; \Gamma \stackrel{i}{\vdash} \mathrm{box}\, M : \Box A} \Box I \qquad \frac{\Psi; \Gamma \stackrel{i}{\vdash} M : \Box A}{\Psi; \Gamma \stackrel{i}{\vdash} \mathrm{unbox}\, M : A} \Box E$$

$$\frac{\Psi; \Delta \stackrel{i}{\vdash} M : \Box A}{\Psi; \Delta; \Gamma \stackrel{i}{\vdash} M : \Box A}\, pop$$

A term M is typed in a *local context* Γ under a stack Ψ. Initially, the stack Ψ and local context Γ would both be empty. Variables are added only to the local context ($\to I$) and may be looked up only in the local context (var). Whenever a *box* operator is encountered ($\Box I$), an empty local context is created and the current one pushed onto the stack. The size of the stack thus increases as we descend into the scopes of nested *box* operators. We can pop the local context from the context stack only if the subterm we are analyzing has a type of the form $\Box A$. The corresponding restriction in the explicit system is that the domain of the substitution σ in the *box* operator must be a context of the form $\Box\Gamma$.

Our presentation of S4 can be seen as a natural deduction analogue of its possible world semantics due to Kripke. Each context Δ of the context stack Ψ represents a possible world; the declarations in the context are the assumptions we make in the corresponding world. The rightmost context represents the current world. New worlds are introduced in the $\Box I$ rule. We can make no assumption about a new world (which is why the new context is initially empty). The $\Box E$ rule represents the reflexivity of the accessiblity relation between worlds: if $\Box A$ holds in the current world, A must also hold in the current world. The *pop* rule implements transitivity, since we can apply it as often as we like to conclude $\Box A$ in any future world given $\Box A$ in the current world. In this sense the system is quite similar to the one given by Martini and Masini [13], except that our formulas and proof terms generically apply to all worlds, while in their system formulas and proof terms uniquely determine their world. Their work shows that we can easily obtain similar formulations of weaker intuistionistic modal logics (such as K or K4) by modifying the $\Box E$ and *pop* rules, but we have not investigated the properties of such systems in our context.

Example 3.1 We verify the following judgment:

$$x{:}B, y{:}B \to \Box A \stackrel{i}{\vdash} \mathrm{box}\,(y\,x) : \Box\Box A.$$

The only applicable rule is $\Box I$. We thus have to show that

$$(x{:}B, y{:}B \to \Box A); \cdot \stackrel{i}{\vdash} y\,x : \Box A.$$

Without any local declarations, the term $y\,x$ can not be typed unless we use the *pop* rule. Its application gives us the subgoal

$$x{:}B, y{:}B \to \Box A \stackrel{i}{\vdash} y\,x : \Box A,$$

which can be trivially verified using $\to E$ and two applications of *var* rule.

Example 3.2 Consider type checking of the term

$$\lambda y{:}\Box B.\ \mathrm{box}\,(\lambda x{:}\Box B \to \Box A.\ \mathrm{box}\,(xy)).$$

The problem of inferring

$$\stackrel{i}{\vdash} \lambda y{:}\Box B.\ \mathrm{box}(\lambda x{:}\Box B \to \Box A.\ \mathrm{box}(xy)) : C_1$$

for a meta-variable C_1 can be straightforwardly reduced to

$$y : \Box B; x : \Box B \to \Box A; \cdot \stackrel{i}{\vdash} xy : C_2,$$

where $C_1 = \Box B \to \Box((\Box B \to \Box A) \to \Box C_2)$ and C_2 is a fresh meta-variable. Since the local context is empty, the rule for application ($\to E$) will eventually fail, leaving us to consider the rule *pop*. It instantiates $C_2 = \Box C_3$ and leads to the subgoal

$$y : \Box B; x : \Box B \to \Box A \stackrel{i}{\vdash} xy : \Box C_3.$$

Now, by using the $\to E$ rule backwards, we obtain subgoals

$$y : \Box B; x : \Box B \to \Box A \stackrel{i}{\vdash} x : C_4 \to \Box C_3 \text{ and } y : \Box B; x : \Box B \to \Box A \stackrel{i}{\vdash} y : C_4.$$

The judgment on the left can be verified by letting $C_4 = \Box B$ and $C_3 = A$, at which point the judgment on the right can also be verified by applying the *pop* rule followed by the *var* rule.

If we compute the value of the meta-variables involved, we see that

$$\stackrel{i}{\vdash} \lambda y{:}\Box B.\ \mathrm{box}(\lambda x{:}\Box B \to \Box A.\ \mathrm{box}(xy)) : \Box B \to \Box((\Box B \to \Box A) \to \Box\Box A).$$

A corresponding explicit term is

$$\lambda y{:}\Box B.\ \mathrm{box}^{y/y}_{y:\Box B} \lambda x{:}\Box B \to \Box A.\ \mathrm{box}^{(xy)/z}_{z:\Box A} z.$$

However such an explicit term is not uniquely determined. For example, the substitutions in the *box* operators might contain extraneous terms.

Example 3.3 We modify the previous example to

$$\lambda y{:}B.\ \mathrm{box}(\lambda x{:}B \to \Box A.\ \mathrm{box}(xy)).$$

If we follow the reasoning above we will see that this term can not be typed and the problem is that

$$y{:}B; x{:}B \to \Box A \stackrel{i}{\vdash} y : B$$

cannot be further reduced, since the subject y does not have a type of the form $\Box C$.

We conclude this section with a few remarks about type checking in $\lambda^{\to\Box}$. Note that the system is not completely syntax-directed due to the non-determinism inherent in the *pop* rule. However, since the type does not change in this rule, uniqueness of types is a trivial property. Moreover, since the *pop* rule decreases the context stack, its number of possible applications is bounded and type checking is decidable.

Theorem 3.4 (Basic Properties of $\lambda^{\to\Box}$)

(i) (Uniqueness of Types) *If* $\Psi;\Gamma \stackrel{i}{\vdash} M : A$ *and* $\Psi;\Gamma \stackrel{i}{\vdash} M : A'$ *then* $A = A'$.

(ii) (Decidability) *Given* $\Psi;\Gamma$ *and* M. *Then it is decidable if there exists a type* A *such that* $\Psi;\Gamma \vdash M : A$. □

The typing rules themselves describe a non-deterministic procedure for type checking when interpreted as a logic program. The remaining non-determinism can be eliminated easily by applying the *pop* rule during search if and only if M contains no variable in the local context Γ free and M has approximate type $\Box A$ (see Section 5.1). For our meta-theoretic investigation of $\lambda^{\to\Box}$ it is simpler not to introduce such an optimization. The problems of type inference or type reconstruction are more difficult and are left to future work.

We can obtain a system in which a valid term uniquely determines its typing derivation by adding a *pop* constructor to terms. Such a system may be useful in situations requiring finer distinctions between proof terms than our system makes, but we have not investigated its properties in depth.

Finally we remark that in the absence of a *box* operator in a term, type checking behaves exactly as in the simply-typed λ-calculus since only the local context will be used.

4 Equivalence between Explicit and Implicit Formulations

After proposing the implicit system we now show its equivalence to the explicit formulation with respect to provability. First, the relationship between explicitly and implicitly typed terms hinted at earlier.

$E \mapsto M$ the explicit term E maps to the implicit term M

$$\frac{}{x \mapsto x} \qquad \frac{E \mapsto M}{\lambda x{:}A.\, E \mapsto \lambda x{:}A.\, M} \qquad \frac{E_1 \mapsto M_1 \qquad E_2 \mapsto M_2}{E_1\, E_2 \mapsto M_1\, M_2}$$

$$\frac{\sigma E \mapsto M}{\mathrm{box}^{\sigma}_{\Box\Gamma} E \mapsto \mathrm{box}\, M} \qquad \frac{E \mapsto M}{\mathrm{unbox}\, E \mapsto \mathrm{unbox}\, M}$$

We note that for any explicit term E there exists a unique implicit term M such that $E \mapsto M$, but not *vice versa*. Corollary 4.3 establishes the equivalence of explicit and implicit formulations, but we must generalize before proving it inductively.

Theorem 4.1 (Soundness) *If* $\Delta_1; \Delta_2; \ldots; \Delta_n; \Gamma \stackrel{i}{\vdash} M : A$ *there exist* σ_1, $\ldots, \sigma_n$, $\Box\Gamma'_1, \ldots, \Box\Gamma'_n$ *and* E *such that:*

(i) $\Box\Gamma'_n, \Gamma \stackrel{e}{\vdash} E : A$;

(ii) $\Box\Gamma'_i, \Delta_{i+1} \stackrel{e}{\vdash} \sigma_{i+1} : \Box\Gamma'_{i+1}, 0 \leq i < n$ *where* $\Box\Gamma'_0 = \cdot$;

(iii) $id_{\Delta_1,\ldots,\Delta_n,\Gamma} \circ (\sigma_1, id_{\Delta_2,\ldots,\Delta_n,\Gamma}) \circ (\sigma_2, id_{\Delta_3,\ldots,\Delta_n,\Gamma}) \circ \ldots \circ (\sigma_n, id_\Gamma)E \mapsto M$.

Proof. By induction on the structure of $\Delta_1; \Delta_2; \ldots; \Delta_n; \Gamma \stackrel{i}{\vdash} M : A$. □

Theorem 4.2 (Completeness) *For all* σ_i, $\Box\Gamma'_i$, Δ_i *where* $1 \leq i \leq n$, Γ, *explicit terms* E *and implicit terms* M, *if*

(i) $\Box\Gamma'_n, \Gamma \stackrel{e}{\vdash} E : A$;

(ii) $\Box\Gamma'_i, \Delta_{i+1} \stackrel{e}{\vdash} \sigma_{i+1} : \Box\Gamma'_{i+1}$, *for* $0 \leq i < n$ *where* $\Box\Gamma'_0 = \cdot$;

(iii) $id_{\Delta_1,\ldots,\Delta_n,\Gamma} \circ (\sigma_1, id_{\Delta_2,\ldots,\Delta_n,\Gamma}) \circ (\sigma_2, id_{\Delta_3,\ldots,\Delta_n,\Gamma}) \circ \ldots \circ (\sigma_n, id_\Gamma)E \mapsto M$.

then $\Delta_1; \Delta_2; \ldots; \Delta_n; \Gamma \stackrel{i}{\vdash} M : A$.

Proof. By induction on the structure of $\Box\Gamma'_n, \Gamma \stackrel{e}{\vdash} E : A$. □

Corollary 4.3 (Correctness of $\lambda^{\rightarrow\Box}$)

(i) (Soundness) *If* $\cdot; \Gamma \stackrel{i}{\vdash} M : A$ *then there exists an* E *such that* $E \mapsto M$ *and* $\Gamma \stackrel{e}{\vdash} E : A$.

(ii) (Completeness) *If* $\Gamma \stackrel{e}{\vdash} E : A$ *and* $E \mapsto M$ *then* $\cdot; \Gamma \stackrel{i}{\vdash} M : A$. □

5 Canonical Forms and Type Preservation

The application of λ-calculi in logical frameworks requires a notion of *canonical form* or *long $\beta\eta$-normal form*, since, typically, the canonical forms of a given type are in 1-1 correspondence with the objects to be represented in a framework (see, for example, [11]). In this section, we introduce the notion of *approximate typing* and prove that every approximately typed term in $\lambda^{\rightarrow\Box}$ is convertible to canonical form. We then show subject reduction and type preservation for (precisely) well-typed terms. In conjunction, this means that any well-typed term can be converted to a well-typed canonical form. We do not prove the strong normalization result which can be obtained by an interpretation into the simply typed λ-calculus. In this section and the remainder of the paper we write $\vdash$ for $\stackrel{i}{\vdash}$, since we deal exclusively with the implicit system.

It is important to keep in mind throughout the development that we do not start from an equational theory (such as $\beta\eta$-conversion), but from an *operational semantics*, presented as an algorithm for conversion to canonical form. Equality is then a defined concept: Two terms are equal if they have the same canonical form. While it is easy to see that a term is $\beta\eta$-equivalent to its canonical form, the admissibility of β and η conversion for equational reasoning is not obvious. We conjecture that a binary logical relations argument along the lines of the one presented below is sufficient to establish this, but we have not checked all details at present.

5.1 Approximate Typing

Unlike in functional programming applications where computation can usually be described independently of types, even the definition of canonical form requires types. In $\lambda^{\to\Box}$, the typing rules are not completely syntax-directed because of the *pop* rule. If we tried to model this faithfully the definition of canonical form and its computation would be complicated and inefficient. Fortunately, once we have checked that a term is well-typed, we can convert it to canonical form without further reference to context stacks or the *pop* rule: It is sufficient to enforce that the term is *approximately typed*. This notion is derived by fusing the context stack into a single context, thereby relaxing the restrictions imposed by the *box* operators. In essence, we allow ourselves to use assumptions from any prior world, not just the current one.

Given a context stack $\Psi = \Delta_1; \ldots; \Delta_n$ we define Ψ^*, the result of *fusing* Ψ into one context by $(\Delta_1; \ldots; \Delta_n)^* = \Delta_1, \ldots, \Delta_n$. The main judgment of approximate typing is

$\Gamma \vdash M \div A$ $\quad M$ has approximate type A in context Γ

$$\frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x \div A}\, var$$

$$\frac{\Gamma, x{:}A \vdash M \div B}{\Gamma \vdash \lambda x{:}A.\, M \div A \to B} \to I \qquad \frac{\Gamma \vdash M \div B \to A \qquad \Gamma \vdash N \div B}{\Gamma \vdash M\, N \div A} \to E$$

$$\frac{\Gamma \vdash M \div A}{\Gamma \vdash \text{box}\, M \div \Box A} \Box I \qquad \frac{\Gamma \vdash M \div \Box A}{\Gamma \vdash \text{unbox}\, M \div A} \Box E$$

Lemma 5.1 (Approximate Typing) *If* $\Psi \vdash M : A$ *then* $\Psi^* \vdash M \div A$.

Proof. By a straightforward induction over the structure of the derivation of $\Psi \vdash M : A$. □

5.2 Canonical Forms

In this subsection, we sketch an argument that shows that every approximately typed term in $\lambda^{\to\Box}$ is convertible to canonical form. Since every well-typed term is also approximately typed, the results will be adequate if we also know conversion to canonical form preserves well-typing in the strict sense. Our proof is based on a Tait-style argument modified to use a Kripke-like interpretation (see [9] for a survey). Ironically, this construction of a Kripke model has nothing to do with Kripke models for S4, but only with Kripke models for intuitionistic logic. In this setting, a world is given by a context Γ, and a future world corresponds to an extended context Γ, Δ. We write $\Gamma' \geq \Gamma$ if Γ' extends Γ with additional declarations.

$\Gamma' \geq \Gamma$ $\quad \Gamma'$ extends Γ.

$$\frac{}{\Gamma \geq \Gamma} \qquad \frac{\Gamma' \geq \Gamma}{\Gamma', x{:}A \geq \Gamma}$$

The definition of canonical forms relies on two, mutually recursive judgments.

$\Gamma \vdash M \Downarrow A$ $\quad M$ is canonical of approximate type A,

$\Gamma \vdash M \downarrow A$ $\quad M$ is atomic of approximate type A.

They are defined by the following inference rules.

$$\frac{\Gamma, x{:}A \vdash M \Downarrow B}{\Gamma \vdash \lambda x{:}A.\, M \Downarrow A \to B} \qquad \frac{\Gamma \vdash M \downarrow b}{\Gamma \vdash M \Downarrow b}$$

$$\frac{\Gamma \vdash M \Downarrow A}{\Gamma \vdash \mathrm{box}\, M \Downarrow \Box A} \qquad \frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x \downarrow A}$$

$$\frac{\Gamma \vdash M \downarrow A \to B \qquad \Gamma \vdash N \Downarrow A}{\Gamma \vdash M\, N \downarrow B} \qquad \frac{\Gamma \vdash M \downarrow \Box A}{\Gamma \vdash \mathrm{unbox}\, M \downarrow A}$$

An alternative characterization of canonical forms in the simply-typed λ-calculus is in terms of *long* $\beta\eta$-normal forms—we will not establish the equivalence to an appropriately generalized notion here. It is easy to see that if $\Gamma \vdash M \Downarrow A$ or $\Gamma \vdash M \downarrow A$ then $\Gamma \vdash M \div A$, but not vice versa.

We would like to establish for arbitrary approximately typed terms that they can be converted into a canonical form according to some algorithm. The judgment for conversion to canonical form given below can be interpreted as an algorithm in the style of logic programming, that is, via a notion of goal-directed search for a term N such that $\Gamma \vdash M \Downarrow N \div A$, if Γ, M and A are given.

$M \xrightarrow{\mathrm{whr}} M'$ $\quad M$ weakly head reduces to M'

$\Gamma \vdash M \Downarrow M' \div A$ $\quad M$ converts to canonical form M' at approximate type A

$\Gamma \vdash M \downarrow M' \div A$ $\quad M$ converts to atomic form M' at approximate type A

$$\frac{}{(\lambda x{:}A.\, M)\, N \xrightarrow{\mathrm{whr}} [N/x]M} \qquad \frac{M \xrightarrow{\mathrm{whr}} M'}{M\, N \xrightarrow{\mathrm{whr}} M'\, N}$$

$$\frac{}{\mathrm{unbox}\,(\mathrm{box}\, M) \xrightarrow{\mathrm{whr}} M} \qquad \frac{M \xrightarrow{\mathrm{whr}} M'}{\mathrm{unbox}\, M \xrightarrow{\mathrm{whr}} \mathrm{unbox}\, M'}$$

$$\frac{\Gamma, x{:}A \vdash M\, x \Downarrow M' \div B}{\Gamma \vdash M \Downarrow \lambda x{:}A.\, M' \div A \to B} \qquad \frac{M \xrightarrow{\mathrm{whr}} M' \qquad \Gamma \vdash M' \Downarrow M'' \div b}{\Gamma \vdash M \Downarrow M'' \div b}$$

$$\frac{\Gamma \vdash \mathrm{unbox}\, M \Downarrow M' \div A}{\Gamma \vdash M \Downarrow \mathrm{box}\, M' \div \Box A} \qquad \frac{\Gamma \vdash M \downarrow M' \div b}{\Gamma \vdash M \Downarrow M' \div b}$$

$$\frac{x{:}A \text{ in } \Gamma}{\Gamma \vdash x \downarrow x \div A} \qquad \frac{\Gamma \vdash M \downarrow M' \div A \to B \qquad \Gamma \vdash N \Downarrow N' \div A}{\Gamma \vdash M\, N \downarrow M'\, N' \div B}$$

$$\frac{\Gamma \vdash M \downarrow M' \div \Box A}{\Gamma \vdash \mathrm{unbox}\, M \downarrow \mathrm{unbox}\, M' \div A}$$

Note that $\Gamma \vdash M \Downarrow M' \div A$ does not imply that M is well-typed or even approximately typed. For example, given a base type b, we have $\Gamma \vdash (\lambda x{:}b.\ \lambda y{:}b.\ y)\, N \Downarrow (\lambda y{:}b.\ y) \div b \to b$ for any object N. However, we have straightforwardly:

Lemma 5.2 (Basic Properties of Conversion)

(i) *If* $\Gamma \vdash M \Downarrow M' \div A$ *then* $\Gamma \vdash M' \Downarrow A$,

(ii) *if* $\Gamma \vdash M \downarrow M' \div A$ *then* $\Gamma \vdash M' \downarrow A$,

(iii) *if* $\Gamma \vdash M \Downarrow M' \div A$ *and* $\Gamma \vdash M \Downarrow M'' \div A$ *then* $M' = M''$, *and*

(iv) *if* $\Gamma \vdash M \downarrow M' \div A'$ *and* $\Gamma \vdash M \downarrow M'' \div A''$ *then* $M' = M''$ *and* $A' = A''$. □

We need a few simple weakening lemmas.

Lemma 5.3 (Weakening) *If* $\Gamma' \geq \Gamma$ *then*

(i) $\Gamma \vdash M \div A$ *implies* $\Gamma' \vdash M \div A$,

(ii) $\Gamma \vdash M \Downarrow M' \div A$ *implies* $\Gamma' \vdash M \Downarrow M' \div A$, *and*

(iii) $\Gamma \vdash M \downarrow M' \div A$ *implies* $\Gamma' \vdash M \downarrow M' \div A$.

Proof. By three straightforward inductions over the structure of derivations. □

Next we define the interpretation that will eventually allow us to show the desired theorem, namely that every approximately typed term is convertible to canonical form. The interpretation is defined under a context Γ, with the intent that $[\![A]\!]$ is a set containing only terms that have approximate type A in the context Γ.

(i) $\Gamma \vdash M \in [\![b]\!]$ iff $\Gamma \vdash M \Downarrow N \div b$ for some N and $\Gamma \vdash M \div b$.

(ii) $\Gamma \vdash M \in [\![A_1 \to A_2]\!]$ iff for every $\Gamma' \geq \Gamma$ and every N, $\Gamma' \vdash N \in [\![A_1]\!]$ implies $\Gamma' \vdash M\, N \in [\![A_2]\!]$.

(iii) $\Gamma \vdash M \in [\![\Box A]\!]$ iff $\Gamma \vdash \mathrm{unbox}\, M \in [\![A]\!]$.

Note that the definition of the interpretation of compound types does not refer to convertibility to canonical form: this is postulated only at base types b. This is a special case of a very general construction of a *logical relation* where we include elements of a desired property (here: convertibility to canonical form) at the level of base types and extend it inductively to the whole type hierarchy. Girard [10] calls the terms in $[\![A]\!]$ *reducible* terms. This is somewhat confusing since it has nothing to do with any underlying notion of reduction, but we will adopt this terminology. Then every reducible term is convertible to canonical form which follows by induction over the structure of types.

Lemma 5.4 (Reducibility Implies Convertibility to Canonical Form)

(i) *If* $\Gamma \vdash M \in [\![A]\!]$ *then* $\Gamma \vdash M \Downarrow M' \div A$ *and* $\Gamma \vdash M \div A$.

(ii) *If* $\Gamma \vdash M \downarrow M' \div A$ *and* $\Gamma \vdash M \div A$ *then* $\Gamma \vdash M \in [\![A]\!]$.

Proof. By induction on the structure of A. □

Lemma 5.5 (Closure under Weak Head Expansion) *If* $\Gamma \vdash M' \in [\![A]\!]$, $M \xrightarrow{\text{whr}} M'$, *and* $\Gamma \vdash M \div A$, *then* $\Gamma \vdash M \in [\![A]\!]$.

Proof. By induction on the structure of A. □

In order to prove Lemma 5.8 by induction we need to generalize to include arbitrary substitution instances. To this end we need to define substitutions and extend the notion of interpretation to substitutions.

Substitutions $\theta ::= \cdot \mid \theta, M/x$

There are approximately and precisely typing variants of substitutions; only the first will be necessary here. For convenience, we nonetheless refer to them as *valid* (rather than approximately valid).

$\Delta \vdash \theta \div \Gamma$ θ is a valid substitution from terms over Γ to terms over Δ.

$$\frac{}{\Delta \vdash \cdot \div \cdot} \qquad \frac{\Delta \vdash \theta \div \Gamma \qquad \Delta \vdash M \div A}{\Delta \vdash (\theta, M/x) \div (\Gamma, x{:}A)}$$

The application of a valid substitution to an approximately typed term is defined as usual and we have the following property.

Lemma 5.6 (Valid Substitution) *If* $\Delta \vdash \theta \div \Gamma$ *and* $\Gamma \vdash M \div A$ *then* $\Delta \vdash \theta M \div A$. □

We extend the interpretation to contexts which will be a set of substitutions. Again, we do not directly refer to the property of convertibility to canonical form, but define it in a general fashion.

(i) $\Delta \vdash \theta \in [\![\cdot]\!]$ *iff* $\theta = \cdot$.

(ii) $\Delta \vdash \theta \in [\![\Gamma, x : A]\!]$ *iff* $\theta = (\theta', M/x)$, $\Delta \vdash \theta' \in [\![\Gamma]\!]$ *and* $\Delta \vdash M \in [\![A]\!]$.

We call substitutions which occur in the interpretations *reducible substitutions*, in analogy with reducible terms. The following lemma is an easy consequence of this definition.

Lemma 5.7 (Validity of Reducible Substitutions) *If* $\Delta \vdash \theta \in [\![\Gamma]\!]$ *then* $\Delta \vdash \theta \div \Gamma$.

Proof. By induction on the structure of contexts. □

We now come to the principal lemma: the reducibility of approximately typed terms under any reducible substitution. The main theorem will follow by using this lemma with the identity substitution.

Lemma 5.8 (Reducibility of Approximately Typed Terms)
If $\Gamma \vdash M \div A$ *and* $\Delta \vdash \theta \in [\![\Gamma]\!]$ *then* $\Delta \vdash \theta M \in [\![A]\!]$.

Proof. By induction on the structure of the derivation of $\Gamma \vdash M \div A$. □

Lemma 5.9 (Reducibility of Identity Substitution) $\Gamma \vdash id_\Gamma \in [\![\Gamma]\!]$

Proof. Let $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$. Then consider $id_\Gamma = x_1/x_1, \ldots, x_n/x_n$. Then $\Gamma \vdash x_i \downarrow x_i \div A_i$ and $\Gamma \vdash x_i \div A_i$ and thus, by Lemma 5.4(2),

$\Gamma \vdash x_i \in [\![A_i]\!]$. Hence, $\Gamma \vdash id_\Gamma \in [\![\Gamma]\!]$. □

Since the identity substitution is reducible we conclude:

Theorem 5.10 (Convertibility to Canonical Form)
If $\Gamma \vdash M \div A$ *then* $\Gamma \vdash M \Downarrow N \div A$ *for some* N.

Proof. By Lemma 5.8 and 5.9, $\Gamma \vdash id_\Gamma M \in [\![A]\!]$. Hence, by Lemma 5.4(1), $\Gamma \vdash M \Downarrow M' \div A$ for some M'. □

5.3 Subject Reduction and Type Preservation

In this section we prove subject reduction and type preservation results. They legitimize the use of approximate typing in the definition of canonical forms and conversion to canonical forms. In terms of functional programming, it means that modal restrictions need to be checked only initially, but not during computation (= conversion to canonical form). Notice that in this subsection we refer to both approximate and precise typing. The proofs require a fusion lemma which allows the fusion of contexts in a context stack, some non-trivial weakening and inversion properties, and a central substitution lemma. They all follow by inductions over the structure of typing derivations and previous lemmas.

Lemma 5.11 (Fusion) *If* $\Delta_1; \ldots; \Delta_i; \Delta_{i+1}; \ldots; \Delta_n \vdash M : A$ *then* $\Delta_1; \ldots; (\Delta_i, \Delta_{i+1}); \ldots; \Delta_n \vdash M : A$ *for* $1 \leq i < n$. □

In addition to weakening in each of the contexts of the context stack, we can also weaken by inserting a whole new modal context into the context stack. However, we may *not* insert it at the end after the last context. For example, $x{:}b \vdash x{:}b$, but $x{:}b; \cdot \not\vdash x{:}b$.

Lemma 5.12 (Modal Weakening)

(i) *If* $\Delta_1; \ldots; \Delta_i; \ldots; \Delta_n \vdash M : A$ *then* $\Delta_1; \ldots; (\Delta_i, x{:}B); \ldots; \Delta_n \vdash M : A$ *for* $1 \leq i \leq n$.

(ii) *If* $\Delta_1; \ldots; \Delta_i; \Delta_{i+1}; \ldots; \Delta_n \vdash M : A$ *then* $\Delta_1; \ldots; \Delta_i; \Delta'; \Delta_{i+1}; \ldots; \Delta_n \vdash M : A$ *for* $1 \leq i < n$. □

Because the rules are no longer fully syntax directed, we also need the following inversion properties. While the first is trivial, the second requires an induction on the derivation of the assumption and modal weakening.

Lemma 5.13 (Inversion)

(i) *If* $\Psi; \Gamma \vdash \lambda x{:}A.\, M : A \to B$ *then* $\Psi; (\Gamma, x{:}A) \vdash M : B$.

(ii) *If* $\Psi; \Gamma \vdash \mathrm{box}\, M : \Box A$ *then* $\Psi; \Gamma; \cdot \vdash M : A$. □

The substitution lemma presents no surprises.

Lemma 5.14 (Substitution)
If $\Delta_1; \ldots; (\Delta_i', x{:}B, \Delta_i''); \ldots; \Delta_n \vdash M : A$ *and* $\Delta_1; \ldots; \Delta_i' \vdash N : B$ *then* $\Delta_1; \ldots; (\Delta_i', \Delta_i''); \ldots; \Delta_n \vdash [N/x]M : A$. □

The following subject reduction theorem applies only to weak head reduction. It can be generalized to redices at arbitrary positions in a term, but this is beside our main interest.

Theorem 5.15 (Subject Reduction) *If* $M \xrightarrow{\text{whr}} M'$ *and* $\Psi \vdash M : A$ *then* $\Psi \vdash M' : A$

Proof. By nested inductions over the derivations of $M \xrightarrow{\text{whr}} M'$ and $\Psi \vdash M : A$, using fusion, modal weakening, inversion, and substitution lemmas. □

The theorem of type preservation finally justifies our use of approximate typing. Note that conversion to canonical form implicitly employs η-expansion, so this does not follow immediately from subject reduction.

Theorem 5.16 (Type Preservation) *Given* $\Psi \vdash M : A$. *Then*

(i) *if* $\Psi^* \vdash M \Downarrow M' \div A$, *then* $\Psi \vdash M' : A$; *and*

(ii) *if* $\Psi^* \vdash M \downarrow M' \div A'$, *then* $A' = A$ *and* $\Psi \vdash M' : A$.

Proof. By mutual inductions over the derivations of $\Psi^* \vdash M \Downarrow M' \div A$ and $\Psi^* \vdash M \downarrow M' \div A'$, with a nested induction over the derivation of $\Psi \vdash M : A$. We also need some lemmas including *approximate typing* (5.1) and *subject reduction* (5.15). □

Corollary 5.17 (Canonical Forms) *If* $\Psi \vdash M : A$ *then there exists an* M' *such that* $\Psi^* \vdash M \Downarrow M' \div A$, $\Psi^* \vdash M' \Downarrow A$, *and* $\Psi \vdash M' : A$. □

Equational reasoning for modal λ-terms can now be achieved by comparing the (unique) canonical forms of two terms for equality modulo α-conversion. We conjecture that β and η conversion between well-typed terms are admissible rules of inference for such an equality.

6 Related Work

As we have pointed out, the explicit system for intuitionistic modal logic has been taken straight from Bierman and de Paiva [5]. A similar system has been developed for linear logic which requires a modality that satisfies the laws of S4 [4].

Our presentation of the implicit system is new as far as we are aware, but some related systems have been given in the literature. A system with explicit quantifiers for worlds has been given by Gabbay and de Queiroz [8]. It also implements a Kripke semantics as a natural deduction system, using explicit quantification over world parameters. This makes it less useful for programming applicatons. Benevides and Maibaum [2] present a formulation where the derivability judgment is explicitly indexed by a world. The same information is contained in the context stack in our presentation. However, their rules for S4 are different in that they do not have an equivalent to the pop rule in its full generality. Instead they add another introduction rule for necessity. Further, they do not introduce a proof term calculus, nor do they study properties such as subject reduction or normalization.

Closest to ours is the study of modal logic by Martini and Masini employing a 2-sequent calculus in its natural deduction form [15,16]. In [13] they present a term assignment system for the system of natural deduction for the intuitionistic modal logics K, K4, KT and S4. In their systems, worlds are also represented structurally as part of the formulation of their judgments. All terms and types are indexed by worlds (addressed by integers) which results in a kind of unicity of worlds and types for valid proof terms.[1] This lends the calculus a certain proof-theoretic elegance, but significantly complicates the meta-theory when compared to our system and is clearly not desirable for practical applications. Martini and Masini study the Church-Rosser property for their system in detail, but not normalization (which they obtain by a simple interpretation into an intuitionistic calculus) or canonical forms.

7 Conclusion and Future Work

We have proposed $\lambda^{\rightarrow\Box}$, a new formulation of a λ-calculus for the intuitionistic modal logic S4. Unlike previous attempts, this formulation does not require substitutions to be embedded in terms or typing derivations. In addition to being more concise, it is well-suited for type checking, hence a significant step forward in applying the notion of modality in areas such as logical frameworks and programming languages. The central idea that enables the type checking procedure is that of a context stack, which groups variables in several different contexts corresponding to the posssible worlds for S4 in Kripke semantics.

Having the application areas in mind, we prove the relevant results. The equivalence to a previous formulation of S4 in which substitutions for modal assumptions are kept explicit gives the correctness of the system with respect to provability; the subject reduction property yields the guarantee of type preservation and the convertibility to canonical form means both the existence of a computation strategy with termination guarantees and the correctness of encoding when the system is used as meta-language in the setting of logical frameworks. A proof of strong normalization is not central to our applications and thus omitted.

Although the system is usable, some intended applications would benefit from further improvements. A type assignment system capable of giving modal types to partially typed or untyped λ-expressions may be desirable, since it reduces the amount of type information that must be provided by a user.

As mentioned before, our calculus lies at the heart of a number of diverse applications in linear logic, logical frameworks and programming languages. We briefly sketch each of them.

Linear logic.

The modal operator ! is required to recover full intuitionistic or classical logic from linear logic. It satisfies the basic laws of S4, but of course interacts

[1] In Section 2.6, they mention, but to not develop the possibility of an implicitly typed version of their system.

with weakening and contraction. A first step in this direction was taken by Martini & Masini [14] who consider a linear logic with a modal operator, but without weakening and contraction. We have designed another system based on similar ideas to the ones described here which permits a simple closed form definition of terms from the ordinary (intuitionistic) λ-calculus. We have established its basic correctness and some normal form properties which will be the subject of a future paper. The connection to Schellinx's *linear decoration* [21] for sequent formulations still remains to be explored.

Modal logical frameworks.

The intuitionistic modal logic S4 can serve as the basis for a modal extension of logical frameworks such as hereditary Harrop formulas [17] used in Isabelle or λProlog. For such an extension to be usable we have to verify *conservative extension* and the *uniform proof property*. Both are corollaries to our canonical form theorem 5.10 and type preservation 5.16. Modal logical frameworks are of interest to explanation-based generalization [7] and may also be used for more concise representations of modal logics themselves.

Binding time analysis.

[6] The most promising application so far is a generalization of Nielson & Nielson's 2-level functional language [18] to admit multiple computation stages, code reuse across various phases of computation, and run-time code generation. This work provides a *logical* explanation and generalization of binding-time analysis within a modal framework. Briefly, each possible world that arises during type checking corresponds to a stage in the computation. Terms of type $\Box A$ are interpreted as *code* of type A, so that a function of type $nat \to \Box(nat \to nat)$ takes a natural number and produces code for the residual function of type $nat \to nat$. The modal restrictions guarantee that we have proper binding-time separation, that is, that all computation connected with the first argument of type nat can in fact be carried out during the first computation stage.

In this application conversion to canonical form is not relevant, since we extend the language with primitive data types and recursion. Instead, we assign an operational semantics to terms in the augmented explicit calculus. The (constructive!) proof of Theorem 4.1 describes a type-directed compilation from the implicit source language to the explicit target language which can then be executed in multiple stages. In the presence of arbitrary recursion, however, this translation is only partially correct and may introduce non-termination, a familiar problem of partial evaluation. Nielson & Nielson's 2-level functional language can be conservatively embedded in the implicit system, which shows that S4 indeed correctly models standard binding-time properties.

Ad hoc polymorphism.

Even if one excludes ad hoc polymorphism from a language like ML, it arises naturally during compilation [12]. Modal *kinds* provide a way to sort out ad hoc and parametric polymorphism in an explicitly polymorphic language. Briefly, $\Box\,\mathit{Type}$ would be the kind of types that may be the subject of a *typecase* statement. Types of this kind thus must be carried at runtime, while types of kind *Type* are compile-time entities and need not be carried at runtime.

Higher-order abstract syntax.

Our original motivation for studying a modal λ-calculus came from the problem of incorporating inductive types, primitive recursion and induction principles into the logical framework LF. Without modal operators this leads to well-known inconsistencies, thus forcing one to choose between induction (as in the Calculus of Inductive Constructions or ALF) and higher-order abstract syntax (as in hereditary Harrop formulas or LF). The rough idea is to decompose the primitive recursive function space $A \Rightarrow B$ into $(\Box A) \rightarrow B$, where $\rightarrow$ defines a parametric function space. This is inspired by a similar decomposition of intuitionistic implication in linear logic. The system given in this paper presents only a small, but, we believe, critical step in this direction. We are currently exploring this in joint work with J. Despeyroux.

In summary, we believe that there are many unexplored opportunities to apply concepts of modal logic in the design of type systems for logical frameworks and programming languages. This paper presents a step towards such applications.

Acknowledgement

We gratefully acknowledge discussions with Gavin Bierman, Rowan Davies, Andrzej Filinski and Simone Martini regarding the subject of this paper. We are also grateful to the anonymous referees for valuable suggestions. Finally we would like to thank Andrzej Filinski for providing an illuminating implementation of the various calculi, translations between them, conversion to canonical forms, and some parts of the correctness proofs in Elf [19], a logic programming language based on the LF logical framework [11]. The implementation is available via anonymous ftp from `ftp.cs.cmu.edu`, file `user/fp/papers/mfps95.tar.Z`.

References

[1] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[2] Mario R. F. Benevides and Thomas S. E. Maibaum. A constructive presentation for the modal connective of necessity. *Journal of Logic and Computation*, 2(1):31–50, 1992.

[3] Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, pages 75–90, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664.

[4] Gavin Bierman. *On Intuitionistic Linear Logic*. PhD thesis, University of Cambridge Computer Laboratory, December 1993. Revised version available as technical report No. 346, August 1994.

[5] Gavin Bierman and Valeria de Paiva. Intuitionistic necessity revisited. In *Proceedings of the Logic at Work Conference*, Amsterdam, Holland, December 1992.

[6] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *Informal Proceedings of the Workshop on Types for Program Analysis*, Aarhus, Denmark, May 1995.

[7] Scott Dietzen and Frank Pfenning. Higher-order and modal logic as a framework for explanation-based generalization. *Machine Learning*, 9:23–55, 1992.

[8] Dov M. Gabbay and Ruy J. G. B. de Queiroz. Extending the Curry-Howard interpretation to linear, relevant and other resource logics. *Journal of Symbolic Logic*, 57(4):1319–1365, December 1992.

[9] Jean H. Gallier. On Girard's "Candidats de Réductibilité". In P. Odifreddi, editor, *Logic and Computer Science*. Academic Press, 1990.

[10] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1989.

[11] 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.

[12] Robert Harper and Greg Morrisett. Compiling polymorphism using intensional type analysis. In Peter Lee, editor, *Proceedings of the 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 130–141, San Francisco, California, January 1995.

[13] Simone Martini and Andrea Masini. A computational interpretation of modal proofs. In H. Wansing, editor, *Proof theory of Modal Logics*. Kluwer, 1994. To appear.

[14] Simone Martini and Andrea Masini. On the fine structure of the exponential rule. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic, Proceedings of the 1993 Workshop on Linear Logic*, Ithaca, New York, 1994. MIT Press. To appear.

[15] Andrea Masini. 2-Sequent calculus: A proof theory of modality. *Annals of Pure and Applied Logic*, 58:229–246, 1992.

[16] Andrea Masini. 2-Sequent calculus: Intuitionism and natural deduction. *Journal of Logic and Computation*, 3:533–562, 1993.

[17] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[18] Flemming Nielson and Hanne Riis Nielson. *Two-Level Functional Languages.* Cambridge University Press, 1992.

[19] 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.

[20] Dag Prawitz. *Natural Deduction.* Almquist & Wiksell, Stockholm, 1965.

[21] Harold Schellinx. *The Noble Art of Linear Decorating.* PhD thesis, Dutch Institute for Logic, Language and Computation, University of Amsterdam, 1994. Available in the ILLC Dissertation Series, 1994-1.

[22] A. S. Troelstra. Natural deduction for intuitionistic linear logic. Unpublished manuscript, May 1993.