A Parametric Fitch-Style Modal Lambda Calculus

Master's thesis in Computer science and engineering

Axel Forsman

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Master's thesis 2023

A Parametric Fitch-Style Modal Lambda Calculus

Axel Forsman

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

A Parametric Fitch-Style Modal Lambda Calculus
Axel Forsman

Supervisor: Nachiappan Valliappan, Department of Computer Science and Engineering
Examiner: Andreas Abel, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

A Parametric Fitch-Style Modal Lambda Calculus
Axel Forsman
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The *necessity modality*, denoted by $\Box$, where the focus lies, has been applied to model staged computations, compartmental purity in functional languages, and more. So called Fitch-style modal deduction, where modalities are eliminated by opening a subproof, and introduced by shutting one, has been adapted for lambda calculi. Different modal logics may be encoded via different open and shut rules. Prior work [1] has given normalization proofs for four Fitch-style formulations of lambda calculi with different modalities, which required repeating the proofs for each individual calculus. A parametric Fitch-style modal lambda calculus generalizing the variants is presented, in order to avoid the repetition and ease further extensions.

Keywords: Necessity, modality, parametric, Fitch, modal, lambda calculus, normalization.

Acknowledgments

I wish to thank my supervisor for their guidance and patience, and my family for their support.

Axel Forsman, Gothenburg, August 30, 2023

Contents

List of Figures

1

Introduction

In this thesis we consider the so called *necessity modality*, and go on to give a parametric modal lambda calculus in the Fitch-style which integrates the necessity modality.

To see the need for modalities in programming languages, it helps to start with understanding the usefulness of type systems. Broadly speaking, type systems are a means of classifying expressions by the values they compute, in order to prove the absence of unsatisfactory program behaviors [2]. If two expressions are known to produce strings of characters, then trying to add them as if they were instead numbers can be statically rejected—without the need to execute the program beforehand. It is clear that a type system cannot be a one-to-one correspondence with the untyped language; that would not make proving anything easier. Designing a type system will therefore always involve trade-offs: Make it more liberal in the amount of programs that are admissible, and reasoning and proving of meta-theoretic properties become harder; do the opposite and there will be more correct programs that are impossible to type check. Thus a plethora of type system concepts have been invented, e.g. union types, algebraic data types, type classes, and polymorphism, to name a few.

Modalities in type theory are one such concept, where they act as unary type constructors that impose some properties and restrictions, i.e. given some type they construct a new type. The necessity modality, denoted by '$\Box$', also mandates the rule of necessitation: It is possible to obtain a value of type $\Box A$ if a value of type A can be derived without any assumptions. It has found use in modeling a number of different things.

In the context of type systems, what is meant by *modeling* some concept is constraining a type in such a way that all values of that type have the set of properties that define the concept. One example of this is the use of modalities for *staged computations* [3], which is further elaborated upon in section 2.3. A staged computation is one that proceeds in a sequence of stages, with the earlier stages generating code for each consecutive stage. By introducing modalities to the type system, we are able to write functions of the type

$$A \to \Box(B \to C)$$

that given a value a of type A will at run-time take the resulting function from B to C, and specialize it depending on the concrete value of a. For example, were we to write

multiplication as a function $\mathbb{N} \to \Box(\mathbb{N} \to \mathbb{N})$, then applying it to the number zero could yield simply the constant function that always returns zero. Here, the '$\Box$' modality is used to model uninterpreted code, and its properties ensure all computation concerned with the argument a is carried out while generating the code for the next stage, else the value a needs to be embedded in the code as a literal. This allows a compiler to safely apply the transformation.

There have been different approaches to integrating the necessity modality into lambda calculus, where we introduce new operators, box and unbox, to work with types involving the $\Box$ type constructor. Fitch-style is one of them, inspired by the proof notation for modal logic due to Fitch [4]. It has been shown that the resulting calculi have great computational properties, qualifying them for adaptation into real-world programming languages. Prior work by Valliappan, Ruch, and Tomé Cortiñas [1] gave formulations of four different Fitch-style modal lambda calculi, each corresponding to different modal logics. They also implemented normalization for these calculi, an important meta-theoretic property. Two of these formulations were proved correct by mechanizing them in the proof assistant Agda [5].

However, there was a lot duplication between the two mechanizations which totaled over five thousand lines of Agda code. This includes the formulation of the simply typed lambda calculus foundation that was present in both. Furthermore, this repetition would only continue to increase with each additional modal lambda calculus mechanized. This in spite of the fact that for all four calculi in question only the rules surrounding the unbox operator vary, while the box operator and everything else stay the same.

The contributions of this thesis are therefore:

- The design of a modal lambda calculus where the unbox operator is parametric, in order to encompass all four modal lambda calculi of interest with a single calculus. With the parameter unifying unbox as a starting point, remaining parameters are identified in order to sufficiently define the calculus.
- A normalization algorithm through the Normalization by Evaluation [6] technique;
- and a proof of its correctness.

The results have been mechanized[1] in Agda, providing a proof of their correctness; most work has gone into writing these proofs. The Agda code uses relatively few abstractions, making it more approachable for those from outside the field.

The parametric calculus is more amenable to language extensions; an extension only has to be implemented and proved correct once for it to apply to all instantiated calculi. Moreover, with all the commonalities of the four calculi extracted, it becomes easier to talk about further generalization to other calculi with additional axioms, such as axiom R. This is discussed in the conclusion.

[1] Available online at `https://github.com/axelf4/pfm-lambda`.

This thesis is structured as follows: chapter 2 contains an overview of the theory underlying the work; and chapter 3 and chapter 4 present the main result of this thesis, the parametric calculus and its accompanying normalization algorithm. In chapter 5 the correctness of normalization is proved; and chapter 6 gives two concrete instantiations of the parametric calculus. Finally chapter 7 concludes by discussing relations to prior work and possible future work.

2

Background

In this chapter we summarize the necessary background knowledge for understanding the rest of the material.

2.1 Modal Natural Deduction

Modalities originate from modal logic. Here we consider the family of modal logics derived from intuitionistic propositional logic extended with the unary connective $\Box$, the inference rule *necessitation*, if A is provable without any premises then so is $\Box A$, and different axioms surrounding $\Box$ [7]. For a formula ϕ, the formula $\Box\phi$ reads "It is necessarily true that ϕ," that is, $\Box$ changes the *mode* of ϕ. The most basic modal logic, IK, comes from the *axiom K* ($\Box(A \supset B) \supset \Box A \supset \Box B$). Axiom K together with *axiom T* ($\Box A \supset A$) gives IT; K and *axiom 4* ($\Box A \supset \Box\Box A$) yield IK4; and K, T and 4 give IS4.

Natural deduction is a proof system wherein reasoning is carried out using "natural" inference rules. Fitch [8] introduced a notation for propositional natural deduction, central to which is the idea of *subordinate proof*. For example, in order to prove "A implies B," $A \supset B$, a subordinate proof may be opened containing the new assumption A, where one sets out to prove B. If successful, the subordinate proof can be shut by introducing $A \supset B$ in the original proof thereby discharging the assumption A.

This may be understood semantically using Kripke's possible worlds interpretation [9], [10], where each world has an assignment of truth-values to each proposition. Opening a subproof means visiting a replica new world, with an increase in knowledge owing to the new assumption, and shutting means returning.

To extend natural deduction to modal logic, Fitch added the notion of a *strict* subordinate proof [4], see the example in Figure 2.1, indicated by a '$\Box$' to the left of the line delineating the subproof. It is differentiated by not introducing a new hypothesis for the antecedent of the implication. Additionally, strict subordinate proofs may access prior derived formulas only of a certain shape from proofs to which they are subordinate. This is in contrast to "ordinary" subordinate proofs wherein any previous formula from a outer proof may be reiterated. This is how the mixing of formulas of different modes is

1	$\Box(A \supset B)$	
2	$\quad \Box A$	
3	$\quad \Box \quad A \supset B$	K-import, 1
4	$\quad \quad A$	K-import, 2
5	$\quad \quad B$	$\supset$-elim, 3, 4
6	$\quad \Box B$	K-export, 3-5
7	$\Box A \supset \Box B$	$\supset$-intro, 2-6
8	$\Box(A \supset B) \supset (\Box A \supset \Box B)$	$\supset$-intro, 1-7

Figure 2.1: Example of a modal natural deduction proof of the formula $\Box(A \supset B) \supset (\Box A \supset \Box B)$. Note how a strict subordinate proof must be opened in order to facilitate reasoning about that inside modalities.

prevented. Their usage is given in the following two rules for the logic K:

K-Import

$$\begin{array}{l} \Box\varphi \\ \vdots \\ \Box \quad \vdots \\ \quad\ \ \varphi \end{array}$$

K-Export

$$\begin{array}{l} \Box \quad \vdots \\ \quad\ \ \varphi \\ \Box\varphi \end{array}$$

With K-Import we may import boxed formulas, but after we are done reasoning about them we must re-box the result with K-Export to make use of it.

2.2 Lambda Calculus

The Curry-Howard isomorphism [11] connects logic and lambda calculus, a bare-bones model of computation consisting of only functions and function application. The grammar of lambda calculus terms is evidently short:

$t, s ::= x$	variable access
$\mid \lambda x.\, t$	lambda abstraction
$\mid t\ s$	function application

The isomorphism states that there is a correspondence between formulae and their proofs in natural deduction, and types and programs of lambda calculus, respectively. Take the formula $A \supset A$ for instance, which says that "A implies A." Proving the

formula is analogous to giving a program of the type $A \to A$, for any type A, and one such program is $\lambda x.\, x$, which implements the identity function.

Now, not all lambda calculus terms are meaningful, e.g. $x\ y$ when x is not a lambda function. Types are one way of allowing evaluation to nevertheless be total, by restricting it to only *well-typed* terms, $\Gamma \vdash t : A$, for some type A. The *simply typed lambda calculus* (STLC) [12], in addition to some set of base types, has a single type constructor $A \to B$ for the types of functions. Then, the *typing relation* $\Gamma \vdash t : A$ is defined in terms of a *typing context* Γ, which is an association of variables and their types, and is used i.a. in the function typing rule: $\lambda x.\, y$ is a well-typed term of type $A \to B$ in context Γ, if y has type B in the context Γ extended with $x : A$.

The proof assistant Agda [5] is itself based on *dependently typed* lambda calculus. "Dependently" meaning definitions of types may depend on values, allowing one to state theorems about the results of functions.

Normalization by Evaluation (NbE) is a technique for reducing lambda calculus terms to their normal forms, which are not further reducible [6]. Instead of implementing the normalization procedure "by hand," you instead proceed by evaluating, before *reifying* the resulting semantic value back into a term. If at any point computation is blocked on a value known only at run time, e.g. on an argument when descending into the body of a lambda abstraction, evaluation proceeds with a so called *neutral value*, containing enough information about its origins to make reification possible.

Modal logic has been adapted for modal lambda calculi [4], where modalities are type constructors that add some properties. In place of the K inference rules, K-Export and K-Import, are two new operators box and unbox respectively. To keep track of subordinate proofs in typing judgments a new structural connective 🔒 is added to the context when a $\Box$ is eliminated, and popped when the subproof is closed.

2.3 Example: Staged Computation

For the twofold purpose of exemplifying the previous section, and showing a practical application of the theory, this section gives an account of the work by Davies and Pfenning [3] on leveraging the modal logic S4 for staged computation.

Consider the problem of run-time code generation due to partial function application. For example, prior to a series of matrix multiplication with the same matrix, the code for the multiplication could be specialized at run-time once the matrix has been computed. If it turns out to contain a lot of zeros then that may be exploited to reduce the amount of computation needed. Detecting where this may be done automatically at compile time is similar to binding-time analysis in partial evaluation [13]. Each subexpression is annotated with its data dependencies to detect whether it may be computed in an early stage after only some function arguments have been applied. However, in practice, automatic choice of where to do run-time specialization may lead to slow-downs

due to the non-negligible cost of code generation. An alternative is to mandate that computation staging be explicitly expressed in the type system.

The insight due to [3] is that run-time code generation demands a quoted source expression, and quotation has been studied in modal logic, with intuitionistic S4 being the logic that models staged computation. We get the following interpretations of modal concepts

- Values of type $\Box A$ are code to be executed in future stage, and compiled to generators for code of type A.
- The rule of necessitation, i.e. we have box $E : \Box A$ if $E : A$ in the empty context, says we may quote any closed expression.
- Axiom T, $\Gamma \vdash \Box A \to A$, is evaluation. Specifically, the unbox constructor on values of type $\Box A$ will do evaluation, and for values of types $\Box \cdots \Box A$ it can be used to splice the quoted expressions into a larger ones, so called *quasiquotation*.
- Axiom 4, $\Gamma \vdash \Box A \to \Box\Box A$, is requotation, producing code that generates the original code.
- In general, we do *not* have $\Gamma \vdash A \to \Box A$, since to quote a function its source code would need to be retrieved which is not possible for arbitrary functions. However, for example integers may be quoted to obtain integer literals, and similarly for Booleans, strings etc.

For a function $A \to \Box B$, the modal type guarantees all computation concerning an argument $a : A$ is done while generating code for a value of type B. Otherwise, to access a in the generated code, it must be manually quoted and inserted into the code. In other words, the argument a exists only in the function body that generates the code, and is not automatically kept around with the generated code.

As an example, multiplication with run-time specialization may be written in this framework as

$$
\begin{aligned}
&\text{mult} : \mathbb{N} \to \Box(\mathbb{N} \to \mathbb{N}) \\
&\text{mult} = \lambda \text{n} : \mathbb{N}.\ \textbf{if}\ \text{n} = 0 \\
&\quad \textbf{then box}\ (\lambda \text{m}.\ 0) \\
&\quad \textbf{else box}\ (\lambda \text{m}.\ \text{m} + \textbf{unbox}\ (\text{mult}\ (\text{n} - 1))\ \text{m})
\end{aligned}
$$

Now multiplication by zero becomes a constant function

$$\mathit{mult}\ 0 \mapsto^* \text{box}\ (\lambda \_.\ 0)$$

and for other values, the entire *mult* function is inlined

$$
\begin{aligned}
\mathit{mult}\ 2 &\mapsto^* \text{box}\ (\lambda x.\ x + (\lambda y.\ y + (\lambda z.\ 0)\ y)\ x) \\
&= \text{box}\ (\lambda x.\ x + (x + 0))
\end{aligned}
$$

2.4 Previous Work

There are at present two main variants of lambda calculi formulations for the necessity modality.

The first variant is the dual-context style proposed by Davies and Pfenning [3] and Pfenning and Wong [14], where two contexts, $\Psi; \Gamma$, are used to keep track of assumptions. The context Ψ holds the assumptions that are "necessarily true," while Γ maintains the true assumptions. The introduction and elimination rules are

$$\frac{\Psi; \cdot \vdash t : A}{\Psi; \Gamma \vdash \text{box } t : \Box A} \qquad \frac{\Psi; \Gamma \vdash t : \Box A \quad \Psi, x : A; \Gamma \vdash s : S}{\Psi; \Gamma \vdash \text{letbox } x = t \text{ in } s : S}$$

The elimination rule destructures $\Box A$, letting its content temporarily be necessarily true in s. Reducing it done by applying a substitution:

$$\text{letbox } x = t \text{ in } s : S \sim s[x \mapsto t]$$

The second variant is Kripke- or Fitch style [3], [7], which models Kripke's possible worlds semantics [9]. The worlds are connected via a *modal accessibility relation*, $\Delta \lhd \Gamma$, which should be construed as saying "the contents of boxed values from the world Δ may be accessed in the future world Γ," as will become apparent upon seeing the corresponding $\Box$-elimination rule. Imposing different properties on the relation yields different modal logics [10]. For example, a reflexive modal accessibility relation corresponds to logic T, a transitive relation, to logic 4, etc.

Kripke style, presented by Davies and Pfenning [3] and Pfenning and Wong [14], represents the previously accessed worlds as a stack of contexts, $\vec{\Gamma} = \Gamma_1; \ldots; \Gamma_n$, as opposed to Fitch style. The $\Box$ introduction and elimination rules look like

$$\frac{\vec{\Gamma}; \cdot \vdash t : A}{\vec{\Gamma} \vdash \text{box } t : \Box A} \qquad \frac{\vec{\Gamma} \vdash t : \Box A \quad |\vec{\Delta}| = n}{\vec{\Gamma}; \vec{\Delta} \vdash \text{unbox}_n \ t : A}$$

where $|\vec{\Delta}|$ is the number of local contexts in the stack $\vec{\Delta}$. The elimination rule takes an integer n called the *modal offset*, encoding how far back into the past to travel to access t of type $\Box A$. Adjusting the allowed values of n gives different modal accessibility relations, and ergo different modal logics.

Hu and Pientka [15] have given a normalization by evaluation proof for Kripke style modal lambda calculi.

Fitch style, on the other hand, its name stemming from Fitch-style natural deduction, comes from Borghuis [4] and Clouston [7]. It differs from Kripke style in that instead of a context stack, a flat list is used, with local contexts delimited by a special '🔒' symbol. The introduction and elimination rules are shown below in chapter 3.

$$\lambda_{\mathrm{IK}}/\Box\text{-Elim}\quad \frac{\Gamma \vdash t : \Box A}{\Gamma, \blacksquare, \Gamma' \vdash \mathrm{unbox}_{\lambda_{\mathrm{IK}}}\, t : A}\ \blacksquare \notin \Gamma' \qquad \lambda_{\mathrm{IT}}/\Box\text{-Elim}\quad \frac{\Gamma \vdash t : \Box A}{\Gamma, \Gamma' \vdash \mathrm{unbox}_{\lambda_{\mathrm{IT}}}\, t : A}\ \#_{\blacksquare}(\Gamma') \leq 1$$

$$\lambda_{\mathrm{IK4}}/\Box\text{-Elim}\quad \frac{\Gamma \vdash t : \Box A}{\Gamma, \blacksquare, \Gamma' \vdash \mathrm{unbox}_{\lambda_{\mathrm{IK4}}}\, t : A} \qquad \lambda_{\mathrm{IS4}}/\Box\text{-Elim}\quad \frac{\Gamma \vdash t : \Box A}{\Gamma, \Gamma' \vdash \mathrm{unbox}_{\lambda_{\mathrm{IS4}}}\, t : A}$$

Figure 2.2: $\Box$-elimination rules for the modal lambda calculi λ_{IK}, λ_{IT}, λ_{IK4} and λ_{IS4} [7].

Valliappan, Ruch, and Tomé Cortiñas [1] implemented normalization for the four modal lambda calculi λ_{IK}, λ_{IT}, λ_{IK4}, λ_{IS4}. As they note, for the four calculi only the $\Box$-elimination rules differ, see figure 2.2. In this thesis, we use that concept to formulate a single parametric calculus generalizing these four calculi, with a $\Box$-elimination rule that is parametric over the concrete modal accessibility relation in question.

3

The Calculus $\lambda_{\mathbf{PFM}}$

In this chapter we give the specification of the simply typed modal lambda calculus λ_{PFM}, which is the main result of this thesis. The calculus is parameterized by the binary relation $\Delta \lhd \Gamma$ on contexts, subject to requirements that will be given below.

Types are constructed out of an uninterpreted base type ι:

$$\textit{Type} \quad A, B ::= \iota \mid A \to B \mid \Box A$$

The base type ι does not stipulate any introduction- nor elimination rules, there simply has to be *some* base type, or else function- and box types would be unrepresentable. Contexts are snoc-lists of types and locks:

$$\textit{Context} \quad \Gamma ::= \cdot \mid \Gamma, A \mid \Gamma, \text{🔒}$$

The intrinsically typed syntax of the language is given in figure 3.1. De Bruijn indices are used to make α-conversion implicit: Instead of variable names, an index at each variable occurrence indicates the nesting depth relative to its corresponding binder, for example

$$\lambda x.\, (\lambda y.\, y\ (\lambda z.\, z))\ (\lambda w.\, x\ w) \iff \lambda.\, (\lambda.\, 0\ (\lambda.\, 0))\ (\lambda.\, 1\ 0)$$

$$\frac{x = |\Gamma'|}{\Gamma, A, \Gamma' \vdash x : A}\ \text{🔒} \notin \Gamma' \ \textsc{Var} \qquad \frac{\Gamma, A \vdash t : B}{\Gamma \vdash \lambda t : A \to B}\ {\to}\textsc{-Intro} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash s : A}{\Gamma \vdash t\ s : B}\ {\to}\textsc{-Elim}$$

$$\frac{\Gamma, \text{🔒} \vdash t : A}{\Gamma \vdash \operatorname{box} t : \Box A}\ \Box\textsc{-Intro} \qquad \frac{\Delta \vdash t : A \qquad \Delta \lhd \Gamma}{\Gamma \vdash \operatorname{unbox} t : A}\ \Box\textsc{-Elim}$$

Figure 3.1: The set of intrinsically typed terms of λ_{PFM}. The modal accessibility relation $\Delta \lhd \Gamma$ is a parameter of the calculus.

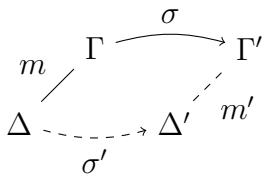

Figure 3.2: Intuition for the contexts involved in the *rewind* operation. We identify an OPE $\Gamma \subseteq \Gamma'$, or substitution $\text{Sub}\,\Gamma\,\Gamma'$, as a map σ on terms from the context Γ to Γ'. Then, given a modal accessibility relation $m : \Delta \lhd \Gamma$, Δ being some past world, $\mathit{rewind}\, m\, \sigma$ yields σ' and m' for some context Δ'.

In order to present the equational theory we define OPE:s and substitutions.

An *order-preserving embedding* (OPE) is a binary relation $\Gamma \subseteq \Delta$ on contexts signifying Γ can be weakened, i.e. add more assumptions, to obtain Δ. It is defined inductively as

$$\frac{}{\text{base} : \cdot \subseteq \cdot} \quad \frac{\Gamma \subseteq \Delta}{\text{weak} : \Gamma \subseteq \Delta, A} \quad \frac{\Gamma \subseteq \Delta}{\text{lift} : \Gamma, A \subseteq \Delta, A} \quad \frac{\Gamma \subseteq \Delta}{\text{lift}_🔒 : \Gamma, 🔒 \subseteq \Delta, 🔒}$$

We define a operation $\text{wk} : \Gamma \subseteq \Delta \to \Gamma \vdash t : A \to \Delta \vdash t : A$ that given an OPE weakens a term. Only the case of weakening an unbox term is unlike the STLC counterpart:

$$\mathit{wk}\, w\ (\text{unbox}\ t\ m) \coloneqq \text{unbox}\ (\mathit{wk}\, w'\ t)\ m' \quad \text{where } m', w' = \mathit{rewind}_{\subseteq}\ m\ w$$

where we require the calculus parameter

$$\mathit{rewind}_{\subseteq} : (m : \Gamma' \lhd \Gamma) \to (w : \Gamma \subseteq \Delta) \to \exists \Delta'.\, \Delta' \lhd \Delta \times \Gamma' \subseteq \Delta'$$

that given a modal accessibility relation m truncates the contexts Γ and Δ in w, see figure 3.2, in order to remove as many locks from both as there are in m. That is to say, it transports w from the future world Γ to instead act on the past world Γ'.

For substitutions—and later environments—we note that both can be seen as replacement lists of items for each type in a context. Thus we choose to define them as concrete instances of a type Rpl_F,[1] parametric over some function $F : \mathit{Type} \to \mathit{Context} \to \mathbf{Set}$ and defined inductively as

$$\frac{}{\cdot : \text{Rpl}_F\, \cdot\, \Delta} \quad \frac{\sigma : \text{Rpl}_F\, \Gamma\, \Delta \qquad x : F\ A\ \Delta}{\sigma, x : \text{Rpl}_F\ (\Gamma, A)\ \Delta} \quad \frac{\sigma : \text{Rpl}_F\, \Gamma\, \Delta \qquad m : \Delta \lhd \Delta'}{\text{lock}\, \sigma\ m : \text{Rpl}_F\ (\Gamma, 🔒)\ \Delta'}$$

This helps unify some of the calculus parameters, and avoids having some parameters depend on e.g. the definition of terms which in turn depends on other parameters. Substitutions may then be defined as $\text{Sub} \coloneqq \text{Rpl}_F$ with $F\ A\ \Gamma \coloneqq \Gamma \vdash A$.

[1] Abbreviation of *Repl*acement, which was chosen as an arbitrary continuation of the sequence of synonyms starting with: Renaming, substitution, ...

$$\beta \text{ equivalence:} \quad \frac{\Gamma, A \vdash t : B \qquad \Gamma \vdash s : A}{(\lambda.\, t)\ s \sim \mathit{subst}\ (\mathit{id}_s, s)\ t} \qquad \frac{\Delta, \blacksquare \vdash t : A \qquad m : \Delta \lhd \Gamma}{\text{unbox}\ (\text{box}\, t)\ m \sim \mathit{subst}\ (\text{lock}\ \mathit{id}_s\ m)\ t}$$

$$\eta \text{ equivalence:} \quad \frac{\Gamma \vdash t : A \to B}{t \sim \lambda.\, (\mathit{wk}\ (\text{weak}\ \mathit{id}_{\subseteq})\ t)\ (\text{var zero})} \qquad \frac{\Gamma \vdash t : \Box A}{t \sim \text{box}\ (\text{unbox}\, t \lhd_{\blacksquare})}$$

Figure 3.3: Equational theory of λ_{PFM}. The rules for lambda abstraction are as for STLC. In the λ-β-conversion rule, a singleton substitution, id_s, s, is applied to t, replacing the zeroth variable with s and decrementing all other de Bruijn indices.

With the exception of the lock constructor the definition of Rpl is as for substitutions in STLC. Adding the alternate constructor $\text{lift}_{\blacksquare} : \text{Rpl}\,\Gamma\ \Delta \to \text{Rpl}\ (\Gamma, \blacksquare)\ (\Delta, \blacksquare)$ to STLC substitutions would allow them to represent local substitutions in any of the "worlds" delimited by locks in the context. Instead, lock with an argument $m : \Delta \lhd \Delta'$ (as used in [1]) makes it possible to unify substitutions and *modal transformations*, where locks are removed and added from contexts as permitted by $(\lhd)$, and which are needed to describe the effect of unboxing.

With this choice of lock we make use of the parameter

$$\lhd_{\blacksquare} : \forall \Gamma.\, \Gamma \lhd \Gamma, \blacksquare$$

in order to be able to define the identity substitution $\mathit{id}_s : \forall \Gamma.\ \text{Sub}\,\Gamma\ \Gamma$.

As for wk in the case of OPE:s, we define $\mathit{subst} : \text{Rpl}\,\Gamma\ \Delta \to \Gamma \vdash t : A \to \Delta \vdash t : A$, using the parameter

$$\mathit{rewind} : (m : \Gamma' \lhd \Gamma) \to (\sigma : \text{Rpl}\,\Gamma\ \Delta) \to \exists \Delta'.\, \Delta' \lhd \Delta \times \text{Rpl}\,\Gamma'\ \Delta'$$

The equational theory of λ_{PFM} is given in figure 3.3, where reflexivity, symmetry, transitivity and congruence rules, which ensure that $(\sim)$ is an equivalence relation, have been omitted. The notation $t \sim s$ says that the terms t and s are equal up to the conversion.

3.1 Summary of Calculus Parameters

Because calculus parameters have been introduced on a per-need basis, this section collects all parameters thus far. These are sufficient for stating the syntax, and the semantics in chapter 4; however, chapter 5 introduces a further set of parameters, restricting the calculi that may be instantiated enough such that it is possible to prove correctness.

- A modal accessibility relation $\Delta \lhd \Gamma$ between contexts is required;

- and a context obtained by appending a new empty local context to some context Γ should be related to Γ:

$$\lhd_{🔒} : \Gamma \lhd (\Gamma, 🔒)$$

- OPE:s can be rewound according to a modal accessibility relation:

$$\mathit{rewind}_{\subseteq} : (m : \Gamma' \lhd \Gamma) \to (w : \Gamma \subseteq \Delta) \to \exists \Delta'.\, \Delta' \lhd \Delta \times \Gamma' \subseteq \Delta'$$

- and similarly for Rpl:s such as substitutions:

$$\mathit{rewind} : (m : \Gamma' \lhd \Gamma) \to (\sigma : \mathrm{Rpl}\, \Gamma\ \Delta) \to \exists \Delta'.\, \Delta' \lhd \Delta \times \mathrm{Rpl}\, \Gamma'\ \Delta'$$

4

Normalization Algorithm

We provide a Normalization by Evaluation algorithm based on a possible worlds model for the parametric calculus in chapter 3. Normal and neutral forms are defined mutually as:

$$\frac{x : \mathrm{Ne}\ \Gamma\ \iota}{\mathrm{ne}\, x : \mathrm{Nf}\ \Gamma\ \iota} \qquad \frac{x : \mathrm{Nf}\ (\Gamma, A)\ B}{\mathrm{abs}\, x : \mathrm{Nf}\ \Gamma\ (A \to B)} \qquad \frac{x : \mathrm{Nf}\ (\Gamma, \text{🔒})\ A}{\mathrm{box}\, x : \mathrm{Nf}\ \Gamma\ (\Box A)}$$

$$\frac{x : A \in \Gamma}{\mathrm{var}\, x : \mathrm{Ne}\ \Gamma\ A} \qquad \frac{x : \mathrm{Ne}\ \Gamma\ (A \to B) \qquad y : \mathrm{Nf}\ \Gamma\ A}{x\ y : \mathrm{Ne}\ \Gamma\ B} \qquad \frac{x : \mathrm{Ne}\ \Delta\ (\Box A) \qquad m : \Delta \lhd \Gamma}{\mathrm{unbox}\, x\ m : \mathrm{Ne}\ \Gamma\ A}$$

The normal forms are β-normal—no β-reductions are possible—and η-long—all variables are maximally applied and unboxed, as the ne constructor only permits neutral values of the base type.

As done in [1], we choose contexts for worlds, OPE:s for the intuitionistic accessibility relation between worlds, and $(\lhd)$ for the modal accessibility relation. (The intuitionistic accessibility relation should be thought of as relating two worlds w_1 and w_2 if w_2 has as much or more knowledge than w_1; for worlds as contexts this means all assumptions in w_1 should be present in w_2 too.) Then we interpret types in the model as

$$\begin{aligned} [\![\iota]\!]_\Gamma &\coloneqq \mathrm{Nf}\ \Gamma\ \iota \\ [\![A \to B]\!]_\Gamma &\coloneqq \forall \Delta.\, \Gamma \subseteq \Delta \to [\![A]\!]_\Delta \to [\![B]\!]_\Delta \\ [\![\Box A]\!]_\Gamma &\coloneqq \forall \Gamma', \Delta.\, \Gamma \subseteq \Gamma' \to \Gamma' \lhd \Delta \to [\![A]\!]_\Delta \end{aligned} \tag{4.1}$$

and contexts as environments, i.e. lists of semantic values, using Rpl,

$$[\![\Gamma]\!]_\Delta \coloneqq \mathrm{Rpl}_{[\![-]\!]}\ \Gamma\ \Delta$$

We have monotonicity for semantic values and environments, i.e. we have $wk_A : \Delta \subseteq \Delta' \to [\![A]\!]_\Delta \to [\![A]\!]'_\Delta$ and $wk_\Gamma : \Delta \subseteq \Delta' \to [\![\Gamma]\!]_\Delta \to [\![\Gamma]\!]'_\Delta$.

The definitions of evaluation, reification and reflection are given in figure 4.1. Variable lookup is as for STLC; the Var rule does not permit access across lock delimiters in the context, thus the lookup function just has to read the variable value from its place in the local environment. The normalization function may then be given as

$$
\begin{array}{lll}
\text{Evaluation} & [\![-]\!] : \Gamma \vdash t : A \to \forall \Delta.\, [\![\Gamma]\!]_\Delta \to [\![A]\!]_\Delta & \\
[\![x]\!]\ \gamma & \coloneqq \text{lookup } x \text{ in } \gamma & \\
[\![\lambda.\, t]\!]\ \gamma & \coloneqq \lambda w\ \hat{a}.\, [\![t]\!]\ (\mathrm{wk}_{\widehat{\Gamma}}\ w\ \gamma, \hat{a}) & \\
[\![t\ s]\!]\ \gamma & \coloneqq [\![t]\!]\ \gamma\ id_\subseteq\ ([\![s]\!]\ \gamma) & \\
[\![\mathrm{box}\, t]\!]\ \gamma & \coloneqq \lambda w\ m.\, [\![t]\!]\ (\mathrm{lock}\ (\mathrm{wk}_{\widehat{\Gamma}}\ w\ \gamma)\ m) & \\
[\![\mathrm{unbox}\, t\ m]\!]\ \gamma & \coloneqq [\![t]\!]\ \gamma'\ id_\subseteq\ m' \quad \text{where } m', \gamma' = \mathit{rewind}\, m\ \gamma & \\
\text{Reification} & \downarrow^A : [\![A]\!]_\Gamma \to \mathrm{Nf}\ \Gamma\ A & \\
\downarrow^\iota\ a & \coloneqq a & \\
\downarrow^{A \to B}\ a & \coloneqq \mathrm{abs}\ (\downarrow^B\ (a\ (\mathrm{weak}\ id_\subseteq)\ (\uparrow^A\ (\mathrm{var}\, \mathrm{zero})))) & \\
\downarrow^{\Box A}\ a & \coloneqq \mathrm{box}\ (\downarrow^A\ (a\ id_\subseteq \lhd_{\text{🔒}})) & \\
\text{Reflection} & \uparrow^A : \mathrm{Ne}\ \Gamma\ A \to [\![A]\!]_\Gamma & \\
\uparrow^\iota\ x & \coloneqq \mathrm{nt}\, x & \\
\uparrow^{A \to B}\ x & \coloneqq \lambda w\ a.\ \uparrow^B\ ((\mathrm{wk}_{\mathrm{Ne}}\, w\ x)\ (\downarrow^A\ a)) & \\
\uparrow^{\Box A}\ x & \coloneqq \lambda w\ m.\ \uparrow^B\ (\mathrm{unbox}\ (\mathrm{wk}_{\mathrm{Ne}}\, w\ x)\ m) &
\end{array}
$$

Figure 4.1: Evaluation, reification and reflection definitions.

Definition 1 (Normalization by Evaluation)**.** Given a term $\Gamma \vdash t : A$, normalization yields a normal form $\mathrm{Nf}\ \Gamma\ A$,

$$nf\, t \coloneqq \downarrow^A\ ([\![t]\!]\ id_e)$$

where id_e is the identity environment, which associates each variable x in Γ with a value given by $\uparrow \mathrm{var}\, x$.

The algorithm can be summarized as follows: Evaluation proceeds as for an interpreter, except closures take an extra OPE argument which allows conjuring fresh variables under binders when reifying functions. Then the resulting semantic value is reified back to its normal form. When reifying a box or function, evaluation resumes with the boxed term, or the closure applied to a neutral form corresponding to the argument type, respectively.

Here the possible worlds inspired interpretation of $\Box A$ has been chosen, instead of the syntax-directed approach of

$$[\![\Box A]\!]_\Gamma = [\![A]\!]_{\Gamma,\text{🔒}}$$

one reason being that the unbox case of evaluation then would require being able to apply the equivalent of a $\mathrm{lock}\, id\ m$ substitution on semantic values, in addition to weakening, whereas currently no such thing is needed, as the m instead goes directly in unbox when reflecting.

5

Correctness

Soundness and completeness of normalization have been proved with respect to possible worlds, which together show that the syntax in chapter 3, and the semantics in chapter 4 are "in agreement".

First off, we define the composition of two Rpl:s, which, while not appearing in the normalization algorithm, is used in the proof of its correctness.

Definition 2 (Composition of Rpl:s)**.** For $\sigma : \mathrm{Rpl}_F\,\Gamma\ \Delta$ and $\delta : \mathrm{Rpl}_G\,\Delta\ \Xi$, F not necessarily equaling G, their composition $\sigma \circ \delta : \mathrm{Rpl}_G\,\Gamma\,\Xi$ is given by

$$\begin{aligned} \cdot \circ \delta &\coloneqq \cdot \\ (\sigma, a) \circ \delta &\coloneqq (\sigma \circ \delta), \mathit{apply}\ \delta\ a \\ \mathrm{lock}\,\sigma\ m \circ \delta &\coloneqq \mathrm{lock}\ (\sigma \circ \delta')\ m' \quad \text{where } m', \delta' = \mathit{rewind}\ m\ \delta \end{aligned}$$

where the definition is generic over a function $\mathit{apply} : \forall A\ \Gamma\ \Delta.\ \mathrm{Rpl}_G\,\Gamma\ \Delta \to F\ A\ \Gamma \to G\ A\ \Delta$. E.g. for composition of substitutions with substitutions *apply* will be *subst*.

As the source context Γ of $\sigma \circ \delta$ is the same as for σ, composition will preserve the inductive shape of σ, but differ i.a. in the $(-,-)$ constructor contents where δ is *apply*:ed.

Next, we introduce the following additional calculus parameters, with which correctness has been proved:

- Rewinding $\mathrm{lock}\,\sigma\ m$ with a modal accessibility relation $\Gamma \lhd \Gamma, 🔒$ should work as expected, i.e. give back m and σ:

  $$\mathit{rewind}\text{-}\lhd_🔒 : \forall m, \sigma.\ \mathit{rewind} \lhd_🔒 (\mathrm{lock}\,\sigma\ m) \equiv m, \sigma$$

  and the same for $\mathit{rewind}_\subseteq$ on $\mathrm{lift}_🔒$.

- The operation *rewind* distributes over composition:

  $$\begin{aligned} \mathit{rewindPres}\text{-}\circ : (m : \Delta \lhd \Gamma) \to (\sigma_1 : \mathrm{Rpl}\,\Gamma\ \Gamma') \to (\sigma_2 : \mathrm{Rpl}\,\Gamma'\ \Gamma'') \\ \to \mathrm{let}\ m', \sigma_1' = \mathit{rewind}\ m\ \sigma_1 \\ \quad\ m'', \sigma_2' = \mathit{rewind}\ m'\ \sigma_2 \\ \quad \mathrm{in}\ \ \mathit{rewind}\ m\ (\sigma_1 \circ \sigma_2) \equiv m'', \sigma_1' \circ \sigma_2' \end{aligned} \tag{5.1}$$

  and likewise for $\mathit{rewind}_\subseteq$.

- *rewind* should preserve identity:

$$\mathit{rewindPresId} : (m : \Delta \lhd \Gamma) \to \mathit{rewind}\, m\ \mathit{id} \equiv m, \mathit{id}$$

  and likewise for $\mathit{rewind}_{\subseteq}$.

  The parameter *rewindPresId* further only has to hold for values of F for which Rpl_F-weakening with the identity OPE is the identity function, i.e.

$$\mathit{wk}_{\mathrm{Rpl}_F}\ \mathit{id}_{\subseteq}\ \sigma = \sigma$$

  for all $\sigma : \mathrm{Rpl}_F\ \Gamma\ \Delta$.

- *rewind* should commute with weakening and substitution composition:

$$\begin{aligned}
&\mathit{rewindWk} && : (m : \Delta \lhd \Gamma) \to (\sigma : \mathrm{Rpl}\,\Gamma\,\Gamma') \to (w : \Gamma' \subseteq \Gamma'') \\
&&& \to \text{let } m', \sigma' = \mathit{rewind}\, m\ \sigma \\
&&&\qquad\ m'', w' = \mathit{rewind}_{\subseteq}\ m'\ w \\
&&&\quad \text{in } \ \mathit{rewind}\, m\ (\mathit{wk}\, w\ \sigma) \equiv m'', \mathit{wk}\, w'\ \sigma' \\
&\mathit{rewindTrim} && : (m : \Delta \lhd \Gamma) \to (w : \Gamma \subseteq \Gamma') \to (\sigma : \mathrm{Rpl}\,\Gamma'\,\Gamma'') \\
&&& \to \text{let } m', w' = \mathit{rewind}_{\subseteq}\ m\ w \\
&&&\qquad\ m'', \sigma' = \mathit{rewind}\, m'\ \sigma \\
&&&\quad \text{in } \ \mathit{rewind}\, m\ (\mathit{trim}\, w\ \sigma) \equiv m'', \mathit{trim}\, w'\ \sigma'
\end{aligned}$$

  where $\mathit{wk} : \Delta \subseteq \Delta' \to \mathrm{Sub}\,\Gamma\ \Delta \to \mathrm{Sub}\,\Gamma\ \Delta'$ and $\mathit{trim} : \Gamma \subseteq \Gamma' \to \mathrm{Sub}\,\Gamma\ \Delta \to \mathrm{Sub}\,\Gamma'\ \Delta$ is substitution and weakening composition and vice versa, respectively.

These enable proving of the necessary weakening and substitution laws, such as associativity of composition, and the fact that *subst* with the identity substitution is the identity function. Indeed, most show up in the goals when proving the unbox cases of the laws.

The reader is reminded that the full details of any proof are available in the provided Agda code.

5.1 Soundness

Soundness states that a term is convertible to its normal form.

Theorem 1 (Soundness)**.** *If* $\Gamma \vdash t : A$*, then* $t \sim \ulcorner \mathit{nf}\, t \urcorner$.

The proof is an extension of the corresponding proof for STLC [16], and established by

a *Kripke logical relation* [17] between terms and semantic values:

$$
\begin{aligned}
(\simeq^A) &\subseteq \Gamma \vdash A \times [\![A]\!]_\Gamma \\
t \simeq^\iota \hat{t} &:= t \sim \ulcorner \hat{t} \urcorner \\
t \simeq^{A \to B} \hat{t} &:= (w : \Gamma \subseteq \Delta) \to \forall a : \Delta \vdash A, \hat{a} : [\![A]\!]_\Delta.\, a \simeq \hat{a} \\
&\qquad \to \mathrm{app}\ (wk\ w\ t)\ a \simeq \hat{t}\ w\ \hat{a} \\
t \simeq^{\Box A} \hat{t} &:= (w : \Gamma \subseteq \Gamma') \to (m : \Gamma' \lhd \Delta) \to \mathrm{unbox}\ (wk\ w\ t)\ m \simeq \hat{t}\ w\ m
\end{aligned}
$$

It is *logical* in the sense that terms and semantic values of box/function types are related iff the results from unboxing/applying both to related terms and values, are related; "Kripke" means that we may first extend the context with a weakening. Notice how the definition of the logical relation has the same shape as that of semantic values, see equation (4.1). We will prove for each inductive step of normalization that $(\simeq)$ is maintained; just as *nf* always returns a normal form through reification, after reification we will get a $(\sim)$ out of $(\simeq)$.

We extend $(\simeq)$ to a relation $(\simeq_{\mathrm{ctx}})$ between substitutions and environments elementwise related by $(\simeq)$, and show that the interpretations of a term in related substitutions and environments are related, the so called *fundamental theorem* of the logical relation:

Lemma 2 (Fundamental theorem)**.** *If* $t : \Gamma \vdash A$*,* $\sigma : \mathrm{Sub}\ \Gamma\ \Delta$*,* $\delta : \mathrm{Env}\ \Gamma\ \Delta$ *and* $\sigma \simeq_{ctx} \delta$*, then* $subst\, \sigma\, t \simeq [\![t]\!]\, \delta$*.*

Proof. By induction on t. For the case of $t = \mathrm{box}\ s$, it needs to be shown that for all $w : \Delta \subseteq \Delta'$, $m : \Delta' \lhd \Xi$,

$$\mathrm{unbox}\ (\mathrm{box}\ (wk\ (\mathrm{lift}_🔒\ w)\ (subst\ (\mathrm{lock}\ \sigma \lhd_🔒)\ s))) \simeq [\![s]\!]\ (\mathrm{lock}\ (wk\ w\ \delta)\ m)$$

The induction hypothesis gives

$$subst\ (\mathrm{lock}\ (wk\ w\ \sigma)\ m)\ s \simeq [\![s]\!]\ (\mathrm{lock}\ (wk\ w\ \delta)\ m)$$

which we compose with a conversion proof on the left

$$
\begin{aligned}
&\Box\text{-}\beta\ (wk\ (\mathrm{lift}_🔒\ w)\ (subst\ (\mathrm{lock}\ \sigma \lhd_🔒)\ s))\ m \\
&\quad : \mathrm{unbox}\ (\mathrm{box}\ (wk\ (\mathrm{lift}_🔒\ w)\ (subst\ (\mathrm{lock}\ \sigma \lhd_🔒)\ s))) \\
&\qquad \sim subst\ (\mathrm{lock}\ id\ m)\ (wk\ (\mathrm{lift}_🔒\ w)\ (subst\ (\mathrm{lock}\ \sigma \lhd_🔒)\ s))
\end{aligned}
$$

Here the term on the right side of $(\sim)$ does not immediately match up with the left side of $(\simeq)$ in the IH, however one can show that they are in fact equal using *rewind-/rewind*$_\subseteq$*-*$\lhd_🔒$ and substitution laws.

For the $t = \mathrm{unbox}\ s\ m$ case, it suffices to rewind the proof of $\sigma \simeq_{\mathrm{ctx}} \delta$ by m to get

$$rewind\, m\, \sigma \simeq_{\mathrm{ctx}} rewind\, m\, \delta$$

and apply the IH on it and s. □

Similarly, we show that reification and reflection also respect $(\simeq)$:

Lemma 3. *Reification and reflection respect* $(\simeq)$*:*

a. If $t : \Gamma \vdash A$, $\hat{t} : [\![A]\!]_\Gamma$ *and* $t \simeq \hat{t}$ *then* $t \sim \ulcorner\downarrow \hat{t}\urcorner$.

b. If $t : \mathrm{Ne}\,\Gamma\ A$ *then* $\ulcorner t\urcorner \simeq \uparrow t$.

Proof. By induction on A. (Only the $\Box A$ case of (a) is shown here, with the rest left to the formalization.) The proof of $t \simeq \hat{t}$ says unboxing of t and $\hat{t}$ respects $(\simeq)$. Choosing $w = id_\subseteq$ and $m = \lhd_🔒$ and applying the IH on the result yields

$$\mathrm{unbox}\ (wk\ id_\subseteq\ t)\ \lhd_🔒 \simeq \ulcorner\downarrow (\hat{t}\ id_\subseteq\ \lhd_🔒)\urcorner$$

where $wk\ id_\subseteq\ t = t$. Combining this with the $\Box$-η conversion rule for t, i.e.

$$t \sim \mathrm{box}\ (\mathrm{unbox}\ t\ \lhd_🔒)$$

using transitivity and congruence under box of the conversion relation gives the goal. □

The fundamental theorem, using id_s and id_e for substitution and environment, may then be combined with the reification lemma to conclude the soundness proof.

One detail remains, however. Depending on how $(\simeq_{\mathrm{ctx}})$ is defined it may not be possible to rewind it. We know only how to rewind OPE:s and Rpl:s, not arbitrary data types, without taking more rewind functions as parameters. Thus we let $(\simeq_{\mathrm{ctx}}) \coloneqq \mathrm{Rpl}_{A \simeq \widehat{A}}$ where

$$A \simeq \widehat{A}\ A\ \Gamma \coloneqq \exists t : \Gamma \vdash A, \hat{t} : [\![A]\!]_\Gamma.\, t \simeq \hat{t}$$

is triplet of a term, a semantic value, and a proof that the two are related. The substitution may be recouped by mapping the Rpl with a function that picks the t out of each $A \simeq \widehat{A}$, likewise for the environment. For the proof of the unbox case of the fundamental theorem, an additional calculus parameter is needed:

- Rewind should commute with map_{Rpl}:

$$\begin{aligned} rewindCommMap : \forall(f : \forall A, \Gamma.\, F\ A\ \Gamma \to G\ A\ \Gamma), m, \sigma : \mathrm{Rpl}_F\ \Gamma\ \Delta.\\ \pi_1\ (rewind\ m\ \sigma) \equiv \pi_1\ (rewind\ m\ (map\ f\ \sigma))\\ \wedge\ map\ f\ (\pi_2\ (rewind\ m\ \sigma)) \equiv \pi_2\ (rewind\ m\ (map\ f\ \sigma)) \end{aligned}$$

  where π_1 and π_2 is the first and second projections of the product type.

The parameter *rewindCommMap* enables us to get the rewound substitution/environment from a rewound proof of $\sigma \simeq_{\mathrm{ctx}} \delta$. It should be noted that we expect the equivalence to hold a priori, since a *rewind* instantiation cannot observe the contents of the parametric Rpl, which is the only thing map_{Rpl} modifies, as explained in "Theorems for free!" [18]. We still need to take it as a parameter, however, in order to convince Agda.

5.2 Completeness

For two convertible terms, completeness expresses that normalization maps them to the same normal form.

Theorem 4 (Completeness)**.** *If $t \sim t'$, then $nf\, t = nf\, t'$.*

The standard technique for proving completeness of STLC [16], [19] uses a so called *presheaf model*, where terms are interpreted as *natural transformations* between semantic contexts and semantic types. Concretely, what needs to be changed in order to ensure our possible worlds model is also a presheaf model, is to add *naturality* conditions to the interpretation of types. Recall the interpretation of box types from equation (4.1):

$$[\![\Box A]\!]_\Gamma \coloneqq \forall \Gamma', \Delta.\, \Gamma \subseteq \Gamma' \to \Gamma' \lhd \Delta \to [\![A]\!]_\Delta$$

We constrain each box value $\hat{a} : [\![\Box A]\!]_\Gamma$ (and similarly for function values), to fulfill for all $w : \Gamma \subseteq \Gamma'$, $m : \Gamma' \lhd \Delta$ and $w' : \Delta \subseteq \Delta'$,

$$\hat{a}\ (w \circ w'')\ m' = wk_{\widehat{A}}\ w'\ (\hat{a}\ w\ m) \quad \text{where } m', w'' = rewind_{\subseteq}\ m\ w'$$

i.e. semantic boxes should commute with weakenings. This allows us to prove naturality for evaluation, reification and reflection.

For proving completeness, since normalization is evaluation followed by reification, it suffices to show the following lemma:

Lemma 5 (Evaluation is sound w.r.t. conversion)**.** *Given two terms $\Gamma \vdash t, t' : A$ such that $t \sim t'$ and an environment $\gamma : \mathrm{Env}\ \Gamma\ \Delta$, we have*

$$[\![t]\!]\ \gamma = [\![t']\!]\ \gamma$$

The proof is by induction on $t \sim t'$. The $\Box$-η equivalence case—where t' is equal to box (unbox $t \lhd_{🔒}$)—follows from naturality of evaluation and *rewind*-$\lhd_{🔒}$. (Though, since box types are interpreted as functions of a weakening and modal accessibility relation, we must first postulate function extensionality in order to only have to show pointwise equality.)

For $\Box$-β equivalence it needs to be shown that

$$[\![t]\!]\ (\mathrm{lock}\ (wk_{\widehat{\Gamma}}\ id_{\subseteq}\ \gamma')\ m') = [\![subst\ (\mathrm{lock}\ \ id_s\ m)\ t]\!]\ \gamma$$

where $m', \gamma' = rewind\ m\ \gamma$. We accomplish this using the fact that evaluating a substituted term is the same as evaluating the raw term in a modified environment, as summarized in the following lemma:

Lemma 6 (Action of evaluation on substituted terms)**.** *If $\Gamma \vdash t : A$, $\sigma : \mathrm{Sub}\ \Gamma\ \Delta$ and $\gamma : \mathrm{Env}\ \Delta\ \Xi$, then*

$$[\![subst\ \sigma\ t]\!]\ \gamma = [\![t]\!]\ ([\![\sigma]\!]\ \gamma)$$

Here the fact that composition between Rpl:s of different types is defined, see definition 2, comes into play, where we let $apply \coloneqq \lambda\gamma\ t.\, [\![t]\!]\ \gamma$ be evaluation in order to compose substitutions and environments, and interpret $[\![\sigma]\!]\ \gamma$ as $\sigma \circ \gamma$. After applying the lemma it remains to show

$$[\![t]\!]\ (\text{lock}\ (wk_{\widehat{\Gamma}}\ id_{\subseteq}\ \gamma')\ m') = [\![t]\!]\ (\text{lock}\ (id_s \circ \gamma')\ m')$$

which follows from $wk_e\ id_{\subseteq}\ \gamma = \gamma = id_s \circ \gamma$.

6

Concrete Instantiations

In this chapter we give instantiations of λ_{PFM} for the two concrete modal lambda calculi highlighted in [1]. It is worth mentioning that the two resulting calculi are correct by construction, due to the correctness proof of the parametric calculus in chapter 5.

6.1 Intuitionistic K

To obtain λ_{IK} we let the modal accessibility relation, $\Delta \lhd_{\lambda_{\text{IK}}} \Gamma$, be as in Figure 6.1, i.e. Γ is an extension of $\Delta, 🔒$ to the right without adding locks. The $\lhd_🔒$ parameter is given by the base case, base, of the relation.

The implementations of $\mathit{rewind}_{\subseteq}$ and rewind by pattern-matching are both straightforward:

$$\begin{array}{llll} \mathit{rewind}_{\subseteq} & m & (\text{weak}\, w) & := \text{snoc}\, m', w' \quad \text{where } m', w' = \mathit{rewind}_{\subseteq}\ m\ w \\ \mathit{rewind}_{\subseteq} & (\text{snoc}\, m) & (\text{lift}\, w) & := \text{snoc}\, m', w' \quad \text{where } m', w' = \mathit{rewind}_{\subseteq}\ m\ w \\ \mathit{rewind}_{\subseteq} & \text{base} & (\text{lift}_🔒\ w) & := \text{base}, w \end{array}$$

and

$$\begin{array}{llll} \mathit{rewind} & \text{base} & (\text{lock}\, \sigma\ m') & := m', \sigma \\ \mathit{rewind} & (\text{snoc}\ m) & (\sigma, \_) & := \mathit{rewind}\, m\ \sigma \end{array} \tag{6.1}$$

where the each inductive step pops one snoc layer from the modal accessibility relation, and OPE or Rpl respectively, in tandem until reaching the base case. The exception is the weak OPE constructor which is skipped past. In the inductive cases of $\mathit{rewind}_{\subseteq}$, the returned m' is built up with snoc, such that for an OPE $\Gamma \subseteq \Delta$, m' will relate the target context Δ and its rewound parallel; for rewind that information is entirely in the lock constructor instead.

$$\text{base} : \Gamma \lhd_{\lambda_{\text{IK}}} \Gamma, 🔒 \qquad \frac{m : \Delta \lhd_{\lambda_{\text{IK}}} \Gamma}{\text{snoc}\, m : \Delta \lhd_{\lambda_{\text{IK}}} \Gamma, A}$$

Figure 6.1: Modal accessibility relation for λ_{IK}.

All the remaining parameters are trivial—it suffices to case split on the possible constructors of $(\lhd_{\lambda_{\text{IK}}})$ and the OPE or Rpl, until Agda confirms the parameter holds definitionally in the base cases, applying the induction hypothesis elsewhere—with the exception of *rewindPresId* where we need an additional lemma:

Lemma 7. *If* $\sigma : \text{Rpl}\,\Gamma\,\Gamma'$ *such that* $wk_{\text{Rpl}}\ id_{\subseteq}\ \sigma = \sigma$*, then for all* $m : \Delta \lhd_{\lambda_{IK}} \Gamma$

$$\mathit{rewind}\, m\ (\mathit{drop}\, \sigma) = \text{snoc}\, m', \sigma' \quad \textit{where}\ m', \sigma' = \mathit{rewind}\, m\ \sigma$$

where $\mathit{drop} \coloneqq wk_{\text{Rpl}}\ (\text{weak}\ \ id_{\subseteq})$ *weakens* σ *by appending an additional assumption to the target context* Γ'*.*

Proof. The λ_{IK} modal accessibility relation m, viewed as a context extension from Δ to Γ, starts with a 🔒 (see the base constructor). The function *drop* only affects the part of σ pertaining to that right of the lock, which will be removed by the *rewind* operation. □

6.2 Intuitionistic S4

For λ_{IS4}, the modal accessibility relation, which has to be reflexive and transitive [10], is succinctly defined as $\Delta \lhd \Gamma \coloneqq \exists \Xi.\, (\Gamma = \Delta, \Xi)$, i.e. the existence of a context extension from Δ to Γ, as done by Valliappan, Ruch, and Tomé Cortiñas [1], except it makes *rewindPresId* impossible to implement without modifications. The reason for this is that the argument made in section 6.1 for the *rewindPresId* parameter of λ_{IK}, does not apply here as the $(\lhd)$ in question does not necessarily start with a 🔒. Instead, their formalization adds an additional conversion rule for λ_{IS4} that we omit

$$\frac{\Delta \vdash t : \Box A \qquad e : \Xi \qquad \Delta, \Xi \lhd \Gamma}{\text{unbox}\ \ t\ (e \circ m) \sim \text{unbox}\ (wk\ (to_{\subseteq}\ e)\ t)\ m}\ \text{SHIFT-unbox}\ \ 🔒 \notin \Xi$$

and requires substitution with the identity substitution to only preserve terms up to conversion, and likewise for the right identity of substitution composition.

The $(\lhd_{\lambda_{\text{IS4}}})$ definition used in this development, see Figure 6.2, is therefore both an opportunity to try an alternative approach, and, in at least one aspect, simpler, due to *rewindPresId* only having to be shown once, instead of being duplicated for both *subst* and substitution composition. It can either be explicitly reflexive (the refl constructor), or any context extension that starts with a 🔒 (the ext constructor). With ext alone it would not be reflexive which, recall, is a defining factor of IS4, thereby the addition of refl.

The implementations of $\mathit{rewind}_{\subseteq}$ and *rewind* are similar to their λ_{IK} counterparts, with

$$\text{refl} : \Gamma \lhd_{\lambda_{\text{IS4}}} \Gamma \qquad \frac{e : \text{Ext}\ (\Delta, \text{🔒})\ \Gamma}{\text{ext}\, e : \Delta \lhd_{\lambda_{\text{IS4}}} \Gamma}$$

$$\text{nil} : \text{Ext}\ \ \Gamma\ \Gamma \qquad \frac{e : \text{Ext}\ \ \Gamma\ \Delta}{\text{snoc}\ \ e : \text{Ext}\ \ \Gamma\ (\Delta, A)} \qquad \frac{e : \text{Ext}\ \ \Gamma\ \Delta}{\text{snoc}_{\text{🔒}}\ e : \text{Ext}\ \ \Gamma\ (\Delta, \text{🔒})}$$

Figure 6.2: Modal accessibility relation for λ_{IS4}. The datatype Ext $\Gamma\ \Delta$ is a constructive proof of the existence of a context extension from Γ to Δ.

the addition of a ext ($\text{snoc}_{\text{🔒}}$ _) case that needs to be handled, i.e. for *rewind*:

$$\begin{aligned} &\mathit{rewind}\,(\text{ext}\,(\text{snoc}_{\text{🔒}}\ m))\,(\text{lock}\ \sigma\ m_2) \\ &\qquad = \mathit{trans}_{\lhd}\ m'\ m_2, \sigma' \quad \text{where } m', \sigma' = \mathit{rewind}\,(\text{ext}\ \ m)\ \sigma \end{aligned}$$

where $\mathit{trans}_{\lhd}$ is ($\lhd_{\lambda_{\text{IS4}}}$) composition. Here, compared to the base case ext nil, which is the same as for λ_{IK} in Equation 6.1, which see, rewinding needs to recurse on σ from the lock constructor, as the modal accessibility relation spans multiple 🔒:s. This gives two modal accessibility relations, m' and m_2, that need to be combined, and complicates subsequent proofs of the parameters *rewindPres*-$\circ$ and *rewindWk* quite a bit, which need a lemma:

Lemma 8 (The *rewind* operation distributes over ($\lhd_{\lambda_{\text{IS4}}}$) composition)**.** *If* $m_1 : \Gamma \lhd \Gamma'$, $m_2 : \Gamma' \lhd \Gamma''$, $\sigma : \text{Rpl}\ \ \Gamma''\ \Delta$ *and*

$$\begin{aligned} m_2', \sigma' &:= \mathit{rewind}\, m_2\ \sigma \\ m_1', \sigma'' &:= \mathit{rewind}\, m_1\ \sigma' \end{aligned}$$

then

$$\mathit{rewind}\,(\mathit{trans}_{\lhd}\ m_1\ m_2)\ \sigma = \mathit{trans}_{\lhd}\ m_1'\ m_2', \sigma''$$

For instance, in the ext ($\text{snoc}_{\text{🔒}}$ _) case of *rewindPres*-$\circ$ (recall its definition in Equation 5.1):

$$\mathit{rewindPres}\text{-}\circ\ \ (\text{ext}\,(\text{snoc}_{\text{🔒}}\ m))\,(\text{lock}\ \ \sigma_1\ m_2)\ \sigma_2 = ?$$

it needs to be shown that

$$\begin{aligned} &\mathit{trans}_{\lhd}\ (\pi_1\ X)\ (\pi_1\ (\mathit{rewind}\, m_2\ \sigma_2)), \pi_2\ X \\ &\quad \stackrel{?}{=} \pi_1\ (\mathit{rewind}\,(\mathit{trans}_{\lhd}\ (\pi_1\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1))\ m_2)\ \sigma_2) \\ &\qquad , \pi_2\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1) \circ \pi_2\ (\mathit{rewind}\,(\mathit{trans}_{\lhd}\ (\pi_1\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1))\ m_2)\ \sigma_2) \end{aligned}$$

where $X := \mathit{rewind}\,(\text{ext}\, m)\ (\sigma_1 \circ \pi_2\ (\mathit{rewind}\, m_2\ \sigma_2))$. Applying the induction hypothesis on X yields the new left-hand side

$$\begin{aligned} &\mathit{trans}_{\lhd}\ (\pi_1\ (\mathit{rewind}\,(\pi_1\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1))\ (\pi_2\ (\mathit{rewind}\, m_2\ \sigma_2))))\ (\pi_1\ (\mathit{rewind}\, m_2\ \sigma_2)) \\ &\quad , \pi_2\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1) \circ \pi_2\ (\mathit{rewind}\,(\pi_1\ (\mathit{rewind}\,(\text{ext}\, m)\ \sigma_1))\ (\pi_2\ (\mathit{rewind}\, m_2\ \sigma_2))) \end{aligned}$$

Here, applying Theorem 8 with $m_1 \coloneqq \pi_1\ (\mathit{rewind}\ (\mathrm{ext}\ \ m)\ \sigma_1)$, $m_2 \coloneqq m_2$ and $\sigma \coloneqq \sigma_2$, in order to move $\mathit{trans}_{\lhd}$ into the *rewind* call and merge the two *rewind*:s right of $(\circ)$, gives the right-hand side.

Note that all λ_{IS4} parameters have been proven in Agda[1], except for *rewindPresId* which thus far only has a proof on paper, due to insufficient time.

[1]See the `add-is4` branch in the Git repository.

7

Conclusion

We have given a parametric Fitch-style modal lambda calculus and shown how to instantiate the calculi λ_{IK} and λ_{IS4}. This was done by making the relation between the contexts involved in the unbox operation a parameter, and proceeding to attempt to prove the usual syntactic lemmas needed to prove correctness of NbE for lambda calculi, adding additional calculus parameters when stuck. Ultimately, the objective of formulating a parametric calculus can be considered to have been met.

An avenue that was initially explored, was requiring $(\lhd)$ to be a context extension, as it is in both instantiations given in chapter 6, and thereby reduce the amount of work needed when instantiating. Instead more of the syntactic lemmas would able to be proved directly in the parametric calculus. For example, it could be done in such a way that the $\lhd_{\mathbf{a}}$, *rewind*-$\lhd_{\mathbf{a}}$ and *rewind*$_{\subseteq}$-$\lhd_{\mathbf{a}}$ parameters become implicit. This was abandoned, however, as the convenience did not outweigh the perceived loss of generality.

The parallel between lambda abstractions and boxes, where similar to how lambdas are applied by supplying an argument value, boxes can be seen as instead being applied with a modal accessibility relation, helped when doing the correctness proofs. Frequently, this symmetry meant cases concerning box and unbox could be somewhat inferred from looking at the equivalent STLC proofs. Therefore the main difficulties were proving the substitution laws while simultaneously formulating the parameters, and choosing a representation of $(\simeq_{\mathrm{ctx}})$ such that it could be rewound.

The Kripke-style calculus and its normalization by evaluation algorithm by Hu and Pientka [15], are undoubtedly similar to the topic of this thesis. Indeed, the differences between Fitch- and Kripke-style calculi are purely syntactical. Still, the previous lack of generality of Fitch-style calculi has been quoted [20] as an apparent disadvantage of Fitch-style compared to Kripke-style; the results in this thesis show that this is not altogether the case. It is worth mentioning that the Fitch-style is closer to the categorical semantics given by Clouston [7]. One has to concede, however, that the Kripke-style formulation is simpler: The modal accessibility relation is represented as a single integral modal offset, instead of a proof witness that is carried around of how the past and future contexts are related.

This raises the question of whether the Fitch-style accessibility relation being more

general can be meaningful in some way. One possible setback of the way things have been done in this thesis, is that the calculus parameter $\lhd_{🔒}$ imposes too strict a requirement on the accessibility relation ($\lhd$). For example, a version of λ_{IS4} where $\Delta \lhd \Gamma$ if removing all 🔒:s that occur in Γ gives Δ—effectively used in [21]—is not an instance of the parametric calculus, instead one would have to also allow e.g. the removal of any number of locks. That said, the current inclusion of $\lhd_{🔒}$ does arise naturally from the need for identity substitutions.

The formalization in chapter 3 made use of the parametric construct Rpl, in order to cut down on repetition. Allais, Chapman, McBride, *et al.* [22] demonstrated how the concept can be taken further. Their result is similar in that renamings, substitutions and semantic environments may be represented uniformly, however they go beyond that by implementing a generic term traversal that may be instantiated to obtain renaming and substituting of terms and normalization by evaluation, i.a. Specially useful is how this allows for succinctly proving fusion lemmas, e.g. that substitution followed by renaming can be subsumed by a single substitution. Compared to Rpl, their definition of generalized substitutions uses a function space instead of a datatype, i.e. substitutions are instances of $\Gamma \ni A \to \Delta \vdash A$, however this poses problems once modalities are introduced. With a datatype, interactions with context locks are limited by what the lock constructor allows. With functions we are always able to give a proof of $\Gamma, 🔒 \ni A \to \Delta \vdash A$ for any Δ, as $\Gamma, 🔒 \ni A$ is uninhabited since no variables exist to the right of the lock. This is problematic when using renamings as context morphisms, since lock weakening is not in general allowed in modal lambda calculi.

A potential for future work is to investigate whether to replace OPE:s with renamings, i.e. substitutions where the replacement terms are variables. Not only would this remove some *rewind*$_{\subseteq}$/*rewind* repetition in the parameters, but it would also be a step toward being able to add the *R axiom* ($A \to \Box A$). Valliappan and Ruch [23] showed that the condition

$$R_m \subseteq R_i$$

on R_i and R_m, the intuitionistic and modal accessibility relation, respectively, is sufficient for ensuring axiom R is satisfied in the model (with the IR Var rule and corresponding OPE definition). Unlike with OPE:s as given here, the condition holds when picking renamings for the intuitionistic accessibility relation, since renamings encapsulate lock weakenings through the lock constructor.

Finally, another thing that was planned in case extra time presented itself by the end of the project, was implementing supplementary language extensions, such as Booleans, and/or pattern matching. Any extension that does not interact with the modal fragment should not pose difficulty, and it would have been nice to exercise this. Unfortunately, there was not enough time for this.

Bibliography

[1] N. Valliappan, F. Ruch, and C. Tomé Cortiñas, "Normalization for fitch-style modal calculi," *Proceedings of the ACM on Programming Languages*, vol. 6, no. ICFP, pp. 772–798, 2022.

[2] B. C. Pierce, *Types and programming languages*. MIT press, 2002.

[3] R. Davies and F. Pfenning, "A modal analysis of staged computation," *Journal of the ACM (JACM)*, vol. 48, no. 3, pp. 555–604, 2001.

[4] V. A. J. Borghuis, "Coming to terms with modal logic: On the interpretation of modalities in typed l-calculus," Ph.D. dissertation, 1994.

[5] U. Norell, *Towards a practical programming language based on dependent type theory*. Chalmers University of Technology, 2007, vol. 32.

[6] U. Berger and H. Schwichtenberg, "An inverse of the evaluation functional for typed lambda-calculus," in *Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Los Alamitos, 1991, pp. 203–211. DOI: 10.1109/LICS.1991.151645.

[7] R. Clouston, "Fitch-style modal lambda calculi," in *Foundations of Software Science and Computation Structures: 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018. Proceedings 21*, Springer, 2018, pp. 258–275.

[8] F. B. Fitch, *Symbolic logic: An introduction*. Ronald Press Co, 1952.

[9] S. A. Kripke, "Semantical analysis of modal logic i. normal modal propositional calculi," *Mathematical Logic Quarterly*, vol. 9, no. 5-6, pp. 67–96, 1963. DOI: 10.1002/malq.19630090502.

[10] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[11] W. A. Howard, "The formulae-as-types notion of construction," *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, vol. 44, pp. 479–490, 1980.

[12] A. Church, "A formulation of the simple theory of types," *The journal of symbolic logic*, vol. 5, no. 2, pp. 56–68, 1940.

[13] M. Leone and P. Lee, "Lightweight run-time code generation.," *PEPM*, vol. 94, pp. 97–106, 1994.

[14] F. Pfenning and H.-C. Wong, "On a modal λ-calculus for s4," *Electronic Notes in Theoretical Computer Science*, vol. 1, pp. 515–534, 1995.

[15] J. Z. Hu and B. Pientka, "A categorical normalization proof for the modal lambda-calculus," *Electronic Notes in Theoretical Informatics and Computer Science*, vol. 1, 2023.

[16] A. Kovács, "A machine-checked correctness proof of normalization by evaluation for simply typed lambda calculus," M.S. thesis, Eötvös Loránd University, Budapest, 2017. [Online]. Available: `https://andraskovacs.github.io/pdfs/mscthesis.pdf`.

[17] G. D. Plotkin, "Lambda definability and logical relations," University of Edinburgh, Memorandum SAI-RM-4, 1973. [Online]. Available: `https://homepages.inf.ed.ac.uk/gdp/publications/logical_relations_1973.pdf`.

[18] P. Wadler, "Theorems for free!" In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, 1989, pp. 347–359.

[19] T. Altenkirch, M. Hofmann, and T. Streicher, "Categorical reconstruction of a reduction free normalization proof," in *Category Theory and Computer Science*, vol. 953, 1995, pp. 182–199.

[20] J. Hu, "Foundations and applications of modal type theories," [Online]. Available: `https://hustmphrrr.github.io/asset/pdf/proposal.pdf`.

[21] D. Gratzer, J. Sterling, and L. Birkedal, "Implementing a modal dependent type theory," *Proceedings of the ACM on Programming Languages*, vol. 3, no. ICFP, pp. 1–29, 2019.

[22] G. Allais, J. Chapman, C. McBride, and J. McKinna, "Type-and-scope safe programs and their proofs," in *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, 2017, pp. 195–207.

[23] N. Valliappan and F. Ruch, "Fitch-style applicative functors (extended abstract)," Draft 2023. [Online]. Available: `https://nachivpn.me/r.pdf` (visited on 04/04/2023).