Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Intuitionistic Modal Logic IS5

Formalizations, Interpretation, Analysis

Agata Murawska

A Master's Thesis written under the supervision of
Małgorzata Biernacka

Wrocław, May 2013

Abstract

IS5 is an intuitionistic variant of S5 modal logic, one of the normal modal logics, with accessibility relation defined as an equivalence. In this thesis we formalize two known variants of natural deduction systems for IS5 along with their corresponding languages. First, the syntactically pure $\mathrm{IS5_{LF}}$ variant that does not name the modal worlds, is due to Galmiche and Sahli. The second one, $\mathrm{IS5_{L}}$, using world names (labels) in inference rules and in terms of its language, is due to Tom Murphy et al.

For each of the languages accompanying these logics we prove standard properties such as progress and preservation. We show the connection between these languages via a series of type-preserving transformations of terms and contexts. Finally, we give a proof of termination for the call-by-name strategy of evaluation using logical relations method.

The formalization of all of the above properties is done in Coq[1] proof assistant. In particular, the proof of termination allows – via Curry-Howard isomorphism – to extract the evaluator in OCaml from the proof.

Our contributions include providing a term language $\mathrm{L_{IS5\text{-}LF}}$ for $\mathrm{IS5_{LF}}$, as well as creating an in-between logic $\mathrm{IS5_{Hyb}}$ and the corresponding language $\mathrm{L_{IS5\text{-}Hyb}}$, used when showing the connection between $\mathrm{L_{IS5\text{-}L}}$ and $\mathrm{L_{IS5\text{-}LF}}$. For the former language we formalize the termination of call-by-name evaluation strategy.

[1]The Coq development is available at author's github page, `http://github.com/Ayertienna/IS5`.

Acknowledgments

I would like to thank my supervisor, Małgorzata Biernacka, for her support and guidance during the time when I was actively working on this thesis and for her patience during the time when I was not. Not only has she guided me towards the (happy) ending, but she has also devoted many hours to review draft versions of this thesis. I could not be more grateful for that.

I probably would not even begin writing the code that created a base for this thesis without the help and encouragement from my friend, Filip Sieczkowski, who guided me through at the beginning, when I was considering the possible approaches to this formalization. His experience with Coq was of immense value at multiple occasions, when there were more questions than answers. I would have most certainly gone mad without his advice, especially in the early stages.

Last but definitely not least I want to thank my partner, Krzysztof Sakwerda, who not only provided moral support throughout this year, but also had the patience to listen to my rants and raves when a proof "just would not go through". On numerous such occasions he would – after listening to my approach to a problematic proof – ask this one simple question that gave me the final idea. At times I wondered if he understands more of my work than I do.

Agata Murawska

Table of Contents

Chapter 1

Introduction

When thinking about truth value of a sentence such as "I am sad", we have to put it in a context. Am I sad now? In general? Or maybe just sometimes? This context – "now", "in general", "sometimes" – can be thought of as a form of truth qualification: "I am always sad", for instance, might lead us to believe that in this particular moment, I am sad as well. The reverse is usually not true – just because I am sad right now, does not make me sad every single day. We can think of "in general", "right now" and other qualifiers as *modals*.

Statements of logic can be qualified in a similar manner: for a logic to be considered modal it has to qualify the truth. Alethic modalities talk about possibility, necessity and impossibility; epistemic logics treat knowledge and belief as qualifiers; deontic modalities use obligation and permission. Such qualifications are not a new concept – modal logic as a discipline of philosophy dates back to ancient Greece [9], to the works of Aristotle with later developments of Diodorus Cronus, Philo the Dialectician and Chrysippus. They were considering, among other, the problem of future contingents – that is, whether one can meaningfully regard future contingents as true or false now, given that the future is still open and undecided. This lead to developments in both modal and temporal logic.

Modality in philosophy was for many years controversial, not widely used and far from a formal discipline. It wasn't until C. I. Lewis's series of articles and finally his book "Symbolic Logic" from 1932, that modern modal logic was born [2]. Lewis introduced five modal systems, S1 through S5, last two of which are examples of what we now consider *normal modal systems*. His approach was to change the definition of implication – making it more in line with its natural interpretation, that one proposition implies the other if the second can be deduced logically from the first. In particular, in Lewis's opinion it should not be the case that $p \to (q \to p)$. Using possibility ($\Diamond$) as a

modal primitive allowed him to redefine implication as $p \to q := \neg\Diamond(p \wedge \neg q)$. He did not yet think of modality as we tend to do nowadays – there was no separation of possibility from propositional rules, as is the case in almost all current formalizations. While we tend to interpret modality as something additional, with its own set of rules, Lewis saw it as embedded in all the definitions and inseparable. Even though his approach did not last, systems he introduced – in particular S4 and S5 – are still subjects of active research.

Just as C.I. Lewis is considered the father of the syntactic tradition, Saul Kripke is the father of modal semantics. In his 1959 article [10] he introduced what is now referred to as *Kripke semantics* or possible worlds semantics, a formal semantics for non-classical logic systems. This allowed modal logics to further develop as a field of formal reasoning.

Even before the formal semantics for modalities were developed, another important concept was introduced by Fitch [6]: intuitionistic modal logic – first defined as modal logic T extended with formula $\forall x, \Box Fx \to \Box\forall x, Fx$ (known as the Barcan formula), with formalization given as both Hilbert-style axiomatization and Gentzen-style sequent calculus. It was not very influential, not nearly as much as the subsequent work in this field by Prior in his "Time and Modality" [17], where MIPQ logic – now known as IS5 – was defined. Another milestone was a PhD thesis of Alex K. Simpson [18], where he analyses proof theory and semantics for a family of intuitionistic modal logics. Simpson's natural deduction systems based on geometric theories were further developed to better reflect the interpretations given to various modal logics, for example by Pfenning and Davies [15] or Galmiche and Sahli [7].

Another concept crucial for this thesis is a correspondence between computer programs and formal proofs known as Curry-Howard isomorphism. It was first observed by Curry in 1934 that axioms of simple intuitionistic implication logic can be viewed as types for combinators. In 1958 Curry extended this observation by noticing that Hilbert-style deduction systems and a fragment of combinatory logic, used as a standard model of computation, are one and the same thing. Finally, in 1969 Howard made the discovery that natural deduction intuitionistic proofs can be interpreted as typed lambda calculus. The relation between programs and proofs did not stop there – increasingly complicated formal systems were being developed, giving birth to the field of modern type theory.

One particularly interesting calculus that uses higher-order logic, created within this trend is Calculus of Constructions [5] – a basis for the Coq automatic prover, where proving theorems means writing programs. An example

of Coq is particularly important to us as we use it for the formalizations of IS5 logic presented in this thesis. That being said, there were of course other languages developed based on this concept, including ones that gave intuitionistic modal logics meaning as programming languages.

One example of such language is ML5 [12], a language developed by Tom Murphy VII et al. as part of the ConCert platform. It is a programming language based on ML with support for parallel computations. The base for its type system is exactly the logic that is of interest to us – IS5. Another variant of IS5 formalization, this time without language accompanying the logic, came from Galmiche and Salhi's article [7] where they gave an alternative decidability result for intuitionistic S5. These two papers form a base for this thesis. They present different approaches, as Galmiche and Salhi do not use labels for their formalism, resulting in syntactically pure logic, whereas Tom Murphy et al. name contexts, thus extending the usual syntax with labels.

1.1 Goals and approach

The natural deduction system that formed the base for Murphy's ML5 was formalized in Twelf [14], but most articles on modal logics lack verifiable implementations in a theorem prover. In particular an alternative natural deduction system using multi-contextual environment [7] does not have a formal implementation. One of our goals is therefore filling that gap using Coq.

As our interest lies more in theoretical results of variants of IS5, rather than practical implementations, we have chosen Lambda 5 with its simple external language [14], rather than full system with configuration, networks and tables. We provide an alternative (Coq, rather than Twelf) implementation of Lambda 5, but also, using similar syntax, extend Galmiche's natural deduction for L_{IS5} with terms. As the resulting language does not use labels to express changing worlds, we also propose a new variant of this system that uses labels for worlds, but is otherwise very similar to label-free version of IS5, in particular in the use of multi-context environment.

With a number of different implementations of IS5 and their corresponding languages, attempting to translate one language into another is a natural problem. Logical systems in all of the formalisms are considered to be variants of the same logic, IS5, thus we expect them to be equivalent. As programming languages, these systems do differ a little, making translation

a less trivial task.

Our new hybrid language, $\mathrm{L}_{\text{IS5-Hyb}}$, is easily embedded in both labeled and label-free implementations ($\mathrm{L}_{\text{IS5-L}}$ and $\mathrm{L}_{\text{IS5-LF}}$, respectively) – in both cases the translation preserves types and reductions. To go in the other direction – that is, from $\mathrm{L}_{\text{IS5-L}}$ to $\mathrm{L}_{\text{IS5-Hyb}}$ or from $\mathrm{L}_{\text{IS5-LF}}$ to $\mathrm{L}_{\text{IS5-Hyb}}$ – proved to be a more difficult and technical problem, to which we only provide a partial solution.

Finally, one of the most important features of a logical programming language is the termination of its evaluation. In languages with solid theoretical foundations and without general recursion it is often the case that we can prove termination of well-typed programs. Languages based on IS5 are, in that respect, no different. We show termination of evaluation for variant of $\mathrm{L}_{\text{IS5-LF}}$ without possibility ($\Diamond$) using standard Tait's method for simply-typed λ-calculus. We then provide extension of this result to a full $\mathrm{L}_{\text{IS5-LF}}$ language.

It is worth noting that from the proof of termination formalized in Coq we can extract the evaluator in OCaml programming language. In fact, such evaluator has been extracted for the $\mathrm{L}_{\text{IS5-LF}}$ without possibility.

1.1.1 Formalization in Coq

The logical (and type) systems and all the theorems mentioned throughout this thesis have been formalized in Coq. We have left implementation details out of the main text; all the remarks on Coq development can be found in Appendix A. One thing to mention is the libraries and λ-calculus formalization that we chose. We use first-order representation, not higher-order abstract syntax, as α-conversion is not done automatically in Coq. We have chosen Charguéraud's metatheory package([1], [4]) and an approach known as locally nameless representation. The idea behind it is a combination of de Bruijn indices used for bound variables and fresh, named binders for free variables. To give an example, an identity function in such representation looks like: $\lambda A.\underline{0}$, rather than $\lambda(v : A).v$, but expressing a hypothesis that is already in a context does not use indices: $\mathtt{hyp}\,\overline{v}$.

Another important difference is cofinite quantification [1] when introducing new variables. An example of this approach is the following inference rule:

$$\frac{\forall v \notin L, (v\colon A)\colon \Gamma \vdash M^v\colon B}{\Gamma \vdash \lambda A.M\colon A \to B}$$

M^v is an operation replacing $\underline{0}$ with $\overline{v}$ in term M. Note that the premise $(v\colon A)\colon \Gamma \vdash M^v\colon B$ is required to hold for *any* v from outside some given set

L – not necessary the set of free variables used in M. We interpret rule $v \notin L$ as freshness and read this condition as "for any v that is fresh enough". We will expand this topic in the Appendix A.

1.2 Plan of the thesis

In Chapter 2 we give formal introduction to modality, intuitionistic modality and – in particular – IS5. For the latter, we describe axiomatization as well as a number of deduction systems, including the one that was developed as part of this thesis. We also expand the topic of applications of IS5, motivating its usage as a type system.

In Chapter 3 we introduce three languages: $\mathrm{L}_{\text{IS5-L}}$, $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$, corresponding to the three variants of natural deduction system for IS5 that were presented in Chapter 2. We justify correctness of these languages by presenting proofs of progress and preservation.

In Chapter 4 we establish relationships between the languages introduced in Chapter 3. This is done via a series of translations $L_{\text{IS5-L}} \leftrightarrow L_{\text{IS5-Hyb}} \leftrightarrow L_{\text{IS5-LF}}$, all of which preserve type inference.

In Chapter 5 we discuss termination results – first for sublanguage of $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$ type, then for complete $\mathrm{L}_{\text{IS5-LF}}$, which requires slightly more complicated approach using continuations.

Finally, Chapter 6 summarizes contributions of this thesis.

Orthogonally to the above mentioned chapters, we discuss implementation details in Appendix A. Its structure corresponds to the structure of chapters, but in there we focus more on the challenges coming from using Coq proof assistant and differences between paper description and actual implementation.

Chapter 2

Intuitionistic modal logic

In this chapter we introduce modal logics and intuitionistic modal logics. Our focus is on particular IS5, for which we present an axiomatization and three variants of a natural deduction system. Finally we provide motivation and means by which IS5 can be used as a type system. This chapter concludes with a short introduction to Curry-Howard isomorphism.

2.1 Modal logic

We begin by giving a general overview of modal logics. Where do we place them in relation to standard logic, be it classical or intuitionistic? How can we motivate the construction? What can it express?

Suppose we have such a simple logic $\mathcal{L}$, non-modal, as of yet. It may be classical or intuitionistic, but we want to consider it a simple one – typically it would be using a single context to store assumptions and it would have $\rightarrow$, possibly along with some other logical connectives like $\vee$ or $\wedge$, $\neg$ etc.
We can think of logic $\mathcal{L}$ as describing a single world, associated with a singular context it uses. What if we have many worlds? In each of them we can use $\mathcal{L}$ – but that is all. However, if these worlds are connected (like in a directed graph), we may want to be able to say something more, express some properties of edges, of moving between worlds.

We can be interested in expressing, for example, that for the current world, wherever we move, we are in a place where ϕ holds. Such a property will be written as $\Box\phi$. This is very much like saying "for all worlds connected to this one, ϕ". We can therefore think of $\Box$ type as a variant of $\forall$ talking about connected worlds.

Where we have $\forall$, it is usually possible to find $\exists$ equivalent as well. To say that there is a world connected to the current one in which ψ holds, we will write $\Diamond\psi$.

These two operators, $\Box$ and $\Diamond$, are what we call *modal operators*. We read $\Box$ as "it is necessary" and $\Diamond$ as "it is possible". There are more possibilities of modal operators – for example in epistemic logics there are modal operators expressing knowledge (K) and belief (B). In our formalizations we will however limit ourselves to $\Box$ and $\Diamond$.

Different modal logics are distinguished based on both the logic underneath modal operators (intuitionistic, classical, linear, etc.) and accessibility relation. Further in this section we will describe axioms of some standard modal logics. As we are more interested in intuitionistic logics rather than classical ones, we will not dwell into details, rather provide axioms which are used to extend standard Hilbert calculus into a system for modal logic (Hilbert-Lewis system). To give the intuitive meaning of connectives, we will use formal semantics usually referred to as Kripke semantics, described below.

For simplicity, we chose logic that has only four operators: $\rightarrow$, $\neg$, $\Box$, $\Diamond$ and propositional variables p, q, ...

2.1.1 Kripke semantics

The notion of logical consequence for modal logics is defined using Kripke semantics.

We begin by defining a *frame*, that is a pair $\mathcal{F} = (W, R)$, where W is a set of worlds and $R \subseteq W \times W$ is an accessibility relation: $(w, v) \in R$ when we can move from world w to world v directly. We may simply think of $\mathcal{F}$ as of a directed graph.
Next we extend such frame $\mathcal{F}$ into a *model* $\mathcal{M} = (W, R, V)$ where V is a function relating worlds from w with sets of propositions which are true in this world. Finally, $\models$, often referred to as *evaluation relation*, is an extension of V to formulas, defined inductively as:

$w \models p$ if and only if $p \in V(w)$

$w \models \neg A$ if and only if $w \not\models A$

$w \models A \rightarrow B$ if and only if $w \not\models A$ or $w \models B$

$w \models \Box A$ if and only if for all $u \in W$ such that $(w, u) \in R$, $u \models A$

$w \models \Diamond A$ if and only if there exists $u \in W$ such that $(w, u) \in R$ and $u \models A$

$w \models \bot$ should never be true

We read $w \models A$ as "A is satisfied in w".

A given formula ϕ is considered *valid*

- in a model ($\mathcal{M} \models \phi$), if it is satisfied in every world of that model, $w \models \phi$ for all $w \in W_{\mathcal{M}}$
- in a frame ($\mathcal{F} \models \phi$), if it is valid for every possible model of that frame (that is for any choice of V), $\mathcal{M} \models \phi$ for all $\mathcal{M}$ extending $\mathcal{F}$

Below are some variants of modal logics where relation R has some general properties, usually referred to as *frame conditions*.

K enforces no conditions on R;

D requires R to be serial, meaning that we always have some world connected to the current one: $\forall_{w \in W} \exists_{v \in W} (w, v) \in R$;

T requires R to be reflexive;

S4 requires R to be reflexive and transitive;

S5 requires R to be reflexive, symmetric and transitive and Euclidean (meaning that $\forall_{w_1, w_2, w_3 \in W} (w_1, w_2) \in R \wedge (w_1, w_3) \in R \rightarrow (w_2, w_3) \in R$ and $\forall_{w_1, w_2, w_3 \in W} (w_1, w_3) \in R \wedge (w_2, w_3) \in R \rightarrow (w_1, w_2) \in R$; transitive and symmetric relation is always Euclidean, Euclidean and reflexive relation is always symmetric and transitive).

2.1.2 Axiomatization

For the previously mentioned logics we also want to have a syntax. We will provide it using Hilbert-style axiomatization, by adding necessitation rule (**N**) and a number of axioms to the original propositional calculus.

Note that in classical modal logics we can define $\Diamond$ in terms of $\Box$: $\Diamond p := \neg \Box \neg p$.

$$\textbf{MP} \frac{A \rightarrow B \quad A}{B} \qquad \textbf{N} \frac{A}{\Box A}$$

Subst: All substitution instances of theorems of propositional calculus

Distribution Axiom K: $\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$

Reflexivity Axiom T: $\Box p \to p$

Axiom S4: $\Box p \to \Box\Box p$

Axiom B: $p \longrightarrow \Box\Diamond p$

Axiom D: $\Box p \to \Diamond p$

Axiom S5: $\Diamond p \to \Box\Diamond p$

From these axioms we can construct a number of *normal modal logics*, starting with K = **K** + **N** + **MP** + **Subst**, which is the smallest normal modal logic. Adding **T** to K gives us T logic, which is more in line with intuition of "if something is necessarily true, than it is true" – this is the logic used as base for Fitch system [6]. An alternative to T is adding **D** to K, resulting in D system, which eliminates "loose ends" - in D, if something is necessary then it is possible, as there always exists at least one world connected to current one.
More complicated logic arises when T is further enhanced with **S4**, resulting in S4 system. Finally, enriching S4 with **S5** results in S5 logic.

The intuitionistic counterparts of the last two systems have interesting applications in programming languages. IS4 (intuitionistic S4) can be used for staged computations, as described in [15]. IS5 on the other hand is a good model for distributed systems. We will describe this last example in more detail in the final section of this chapter. Before that, however, we need to add intuitionism to modality.

2.2 Intuitionistic modality

As we have already discussed modality in general, we now need to understand how intuitionistic logics differ from classical ones. What changes is the definition of true statements. In classical logics we know that every statement is either true or false. In intuitionistic logics however, a statement is true only if there is a constructive proof that it is true and – dually – it is false only if there is a constructive proof that it is false.
For example in axiomatic calculi for intuitionistic logic we remove from classical axiomatization the law of excluded middle ($p \vee \neg p$) and double negation elimination ($\neg\neg p \to p$) along with any other axioms that would allow us to prove any of these two (e.g. Peirce's law $((p \to q) \to p) \to p$).

Knowing this, we should ask ourselves a question: how to create an intuitionistic analogue of a given standard logic?
What is definite and beyond question is that, just as classical modal logic requires its propositional fragment to be classical logic, intuitionistic modal logic $\mathcal{IML}$ requires such fragment to be intuitionistic logic ($\mathcal{IL}$).
In addition, every instance of a theorem in $\mathcal{IL}$ should be a theorem in $\mathcal{IML}$.
We also expect that adding the law of excluded middle (or double negation elimination etc.) to any variant of $\mathcal{IML}$ results in the appropriate classical counterpart.

This is all very natural, but we have hardly touched on the topic of modal operators. We require them to be independent – meaning that no theorems such as $\Box A \leftrightarrow \neg\Diamond\neg A$ can hold, just as they do not hold for $\vee$ and $\wedge$ in intuitionistic propositional logic.

2.2.1 Kripke semantics

Similarly to classical modal logics, we can define Kripke models for intuitionistic ones. The definitions are combination of Kripke semantics for intuitionistic logic (without modality) and for modal logic. We have already given Kripke semantics for modal logic in the previous section – now we will give them for intuitionistic logic ($\mathcal{IL}$).

Kripke semantics for $\mathcal{IL}$

An intuitionistic Kripke model is a tuple $\mathcal{M} = (W, \leq, V)$, where W is a set of Kripke worlds and $\leq$ is a pre-order on W. V is then a function from worlds to sets of propositions such that for $w \leq w'$, $V_w(p) \subseteq V_{w'}(p)$. Based on V we create a satisfaction relation $\models$ parametrized by Kripke world w:

$w \models p$ if and only if $p \in V(w)$

$w \models A \to B$ if and only if for all $u \geq w$, $u \models A$ implies $u \models B$

$w \models \bot$ should never be true

Note that typically $\neg A$ is defined in terms of $\to$ and $\bot$ as: $\neg A := A \to \bot$

Such semantics use worlds as states of knowledge – the $w \leq w'$ relation between them intuitively expresses that w' is an extension of w. In particular, it should be the case that all the knowledge from w is preserved in w'. This is captured in the following lemma:

Lemma 2.2.1 (Monotonicity)**.** If $w \leq$ w' and $w \models \phi$ then $w' \models \phi$.

Proof. Proof is by induction on the structure of ϕ.

Kripke semantics for $\mathcal{IML}$

As we have mentioned, semantics for $\mathcal{IML}$ combines definitions for modal and intuitionistic logic. We have two types of worlds - modal and Kripke. Kripke worlds (states[1]) are exactly the same as in $\mathcal{IL}$; for a given state w we will have a set of modal worlds known in this state, a relation between these modal worlds and a function evaluating propositions.

W as previously is a set of states. Same as in $\mathcal{IL}$, we require it to be partially ordered using $\leq$ relation

$\{D_w\}_{w \in W}$ gives worlds known in the given state; for $w \in W$, D_w is a non-empty set of modal worlds such that $w \leq w'$ implies $D_w \subseteq D_{w'}$

$\{R_w\}_{w \in W}$ is a family of relations on $D_w \times D_w$, defining modal world accessibility and such that $w \leq w'$ implies $R_w \subseteq R_{w'}$

$\{V_w\}_{w \in W}$ for each $w \in W$, V_w is a function accepting modal worlds from D_w and returning a set of propositions which are true in this modal world; if $w \leq w'$ then $V_w(p) \subseteq V_{w'}(p)$

These form a *birelation model* $\mathcal{B} = (W_{\leq}, \{R_w\}_{w \in W}, \{D_w\}_{w \in W}, \{V_w\}_{w \in W})$ We then define $\models$ as follows:

$w, d \models p$ if and only if $p \in V_w(d)$

$w, d \models A \to B$ if and only if for all $w' \in W$ such that $w' \geq w$, $w', d \models A$ implies $w', d \models B$

$w, d \models \Box A$ if and only if for all $w' \geq w$, for all $d' \in D_{w'}$, if $R_{w'}(d, d')$ then $w', d' \models A$

$w, d \models \Diamond A$ if and only if there exists $d' \in D_w$ such that $R_w(d, d')$ and $w, d' \models A$

Again, there may be frame conditions on R_w that lead to semantics of certain logics like IK, IT, IS4 or IS5.

Lemma 2.2.2 (Monotonicity)**.** If $w \leq$ w' and $w, d \models \phi$ then $w', d \models \phi$.

Proof. Induction on the structure of ϕ.

[1]It may be confusing to use "worlds" both for members of W – Kripke worlds, and for modal worlds known at a given Kripke world. We will therefore call Kripke worlds "states" to avoid confusion.

2.2.2 Axiomatization of IK

We will finish description of intuitionistic modal logics by presenting an axiomatization of IK as per Plotkin's 1986 article [16]. In the next section we will extend this axiomatization into one for IS5.

An axiomatization of IK is the following:

All substitution instances of theorems of intuitionistic logic

$\Box(A \longrightarrow B) \to (\Box A \to \Box B)$ – distribution axiom **K**

$\Box(A \to B) \to (\Diamond A \to \Diamond B)$ – another variant of distribution

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

Along with two standard rules of inference:

$$\mathbf{MP}\ \frac{A \to B \quad A}{B} \qquad \mathbf{N}\ \frac{A}{\Box A}$$

2.3 Formalizations of IS5

The accessibility relation in IS5 is, same as in S5, an equivalence. We will look at only one connected component at a time, as it makes no change in expressive power – the only operators that use worlds at all are $\Box$ and $\Diamond$ and they are limited to connected worlds. With such assumption we do not need to mention accessibility relation at all – as all the worlds in W are connected to each other.

For IS5 we would like to look at several different formalizations. We are also interested in limits of expressive power in this logic – i.e. what cannot we prove in IS5? What can we prove, that we couldn't in weaker logics like IS4?

2.3.1 Expressive power and limitations

When thinking about limitations of IS5, we observe that they either come from the fact that IS5 is intuitionistic or from the expectations that we set for modal logics in general.

An example of the first category can be $\nvdash \Box A \leftrightarrow \neg\Diamond\neg A$, lack of explicit connection between $\Diamond$ an $\Box$ mentioned in one of the previous sections. Note that this formula requires a classical axiom to prove and that it holds in S5.

Second type of limitations are for example $\nvdash A \to \Box A$ and $\nvdash \Diamond A \to A$. These express different strength of three statements: $\Box A$ is stronger than A (but we do not want it to be so strong as to never be true), A is stronger than $\Diamond A$ (but again, $\Diamond A$ should not be always true).
These two statements, along with $\vdash A \to \Diamond A$ and $\vdash \Box A \to A$ capture exactly what Łukasiewicz expected modal operators to behave – however he also expected them to be interdefinable. Prior [17] makes this definition more liberal, excluding the interdefinability, thus defining the intuitionistic variant.

As to expressive power, the following formulas are true in IS5, but not in IS4:

$\vdash \Diamond A \to \Box \Diamond A$

$\vdash \Diamond \Box A \to \Box A$

They are both variants of axiom **S5**. The first one reads "if A is possible, then it is necessarily possible", the second "if A is possibly necessary, then it is necessary".

2.3.2 Axiomatization

An axiomatization of IS5 extends IK with the following axioms:

$\Box A \to A$ – reflexivity axiom **T**

$A \to \Diamond A$ – an instance of reflexivity axiom; "if something is true, then it is possible"

$\Box A \to \Box \Box A$– **S4**, guaranteeing transitivity

$\Diamond \Diamond A \to \Diamond A$ – a variant of **S4** for $\Diamond$

$\Diamond A \to \Box \Diamond A$ – axiom **S5**, stating that the relation is Euclidean

$\Diamond \Box A \to \Box A$ – a variant of **S5**

We can observe that, from the point of view of the axioms, IS5 is to S5 just as IK is to K. Next we want to ensure that this axiomatization matches the intuitive meaning of IS5.

Soundness and Completeness

We want to ensure that the system we have just given is sound and complete. Soundness of a system means that its inference rules provide only valid formulas, where validity of a formula is determined by Kripke semantics[2].

Theorem 2.1 (Soundness)**.** For every theorem ϕ of IS5, $\models \phi$.

Proof. We will validate only new axioms added to IK in the previous section. For proof of IK soundness please refer to [18].

- $\Box A \to A$: We want to justify $w, d \models \Box A \to A$ for any given w, d. By definition this requires that for any $w' \geq w$, $w', d \models \Box A$ implies $w', d \models A$. By definition of $w', d \models \Box A$, for all $w'' \geq w'$ and for all $d' \in D_{w'}$, if $R_{w'}(d, d')$ then $w', d' \models A$. From that we want to conclude $w', d \models A$. Note that as our relation is an equivalence, we can take $d' = d$. Therefore by definition of $w', d \models \Box A$ we can conclude that $w', d \models A$.

- $A \to \Diamond A$: A similar argument is used; this time we want to exists such $d' \in D_w$ $R_w(d, d')$ and $w, d' \models A$. $d' = d$ (in which $w, d \models A$) satisfies both of these conditions.

- $\Box A \to \Box\Box A$: Having $w, d \models \Box A$ we want to show that $w, d \models \Box\Box A$. The latter requires that for all $w' \geq w$, for all $d \in D_{w'}$, if $R_{w'}(d, d')$ then $w', d' \models \Box A$. Let us take any such w' and d'. We want to argue that $w', d' \models \Box A$. This requires that for all $w'' \geq w'$, for all $d'' \in Dw''$, if $R_{w'}(d', d'')$ then $w'', d'' \models A$. From definition of $w, d \models \Box A$, the fact that $\geq$ is a preorder and that $D_{w'} \subseteq D_{w''}$ we have that $w'', d'' \models A$.

- $\Diamond\Diamond A \to \Diamond A$: Dually to the previous proof.

- $\Diamond A \to \Box\Diamond A$: We want to argue that if for all $w' \geq w$, if $w', d \models \Diamond A$ then $w', d \models \Box\Diamond A$. The former means that there exists $d \in D_{w'}$ such that $R_{w'}(d, d')$ and $w', d' \models A$. Let us denote such existing d' as e. We have to conclude from that $w', d \models \Box\Diamond A$, meaning that for $w'' \geq w'$, $d' \in D_{w''}$, if $R_{w''}(d, d')$ then $w'', d' \models \Diamond A$. Now we can use e as the modal world that the definition demands. indeed in w'', d' and e are connected (from $R_{w'}(d, e)$ and properties of the relation R_w) and $w'', e \models A$ follows from monotonicity lemma.

- $\Diamond\Box A \to \Box A$: Analogously as before.

[2]From this moment on, when talking about Kripke semantics we will specifically mean semantics for IS5 – where R has to be an equivalence relation.

Completeness provides the opposite – everything that is considered valid under certain conditions, should also be syntactically derivable.

Theorem 2.2 (Completeness)**.** If $\models \phi$ then ϕ is derivable.

The proof can be found e.g. in Simpson's PhD thesis [18].

2.3.3 Natural deduction

Having axiomatization we can now move to natural deduction, a formalism in which it is easy to use proof-theoretic techniques when showing properties of the logic. We present here three natural deduction systems for IS5. The first one is $\mathrm{IS5_L}$, labeled logic, as described in [14] and [13]. It makes explicit use of worlds, but avoids mentioning relation R between them completely. This is possible as all the worlds are connected to each other, so $R(w, w')$ holds for any pair of worlds w, w'.

One significant difference between $\mathrm{IS5_L}$ and any generic formalizations of intuitionistic modal logics, such as by Simpson [18] is that most of the rules – with just three exceptions – act locally: both premises and the conclusion use the same world. This is motivated by the aimed application of this particular logic – we will talk more about it in the next section and the subsequent chapter. Two of the exceptions we have just mentioned are operations referred to as fetch and get. They act on modal connectives, $\Box$ and $\Diamond$ respectively, and do not change the propositions, but rather the world in which the proof is. The last exception is $\Box$-introduction rule, which requires something to be true in a fresh world in order for it to be universally true.

The complete $\mathcal{ND}$ system for IS5 in labeled variant is presented below. The following notations are used:

Ω denotes the set of known worlds; it has the same role as W in Kripke models

Γ is a context, containing all assumptions. As the assumptions are being made in a specific world, this world's name is also part of the assumption

$A@w$ points to the world, in which A holds – in this case, w

IS5$_{\mathbf{L}}$

$$\texttt{(hyp)}\ \frac{w \in \Omega \qquad (A@w) \in \Gamma}{\Omega; \Gamma \vdash A@w}$$

$$(\to I)\ \frac{\Omega; (A@w) : \Gamma \vdash B@w}{\Omega; \Gamma \vdash (A \to B)@w} \qquad (\to E)\ \frac{\Omega; \Gamma \vdash (A \to B)@w \qquad \Omega; \Gamma \vdash A@w}{\Omega; \Gamma \vdash B@w}$$

$$(\Box I)\ \frac{w \in \Omega \qquad \text{fresh } w_0 \qquad w_0 : \Omega; \Gamma \vdash A@w_0}{\Omega; \Gamma \vdash \Box A@w} \qquad (\Box E)\ \frac{\Omega; \Gamma \vdash \Box A@w}{\Omega; \Gamma \vdash A@w}$$

$$(\Diamond I)\ \frac{\Omega; \Gamma \vdash A@w}{\Omega; \Gamma \vdash \Diamond A@w} \qquad (\Diamond E)\ \frac{\Omega; \Gamma \vdash \Diamond A@w \quad \text{fresh } w_0 \quad w_0 : \Omega; (A@w_0) : \Gamma \vdash B@w}{\Omega; \Gamma \vdash B@w}$$

$$\texttt{(fetch)}\ \frac{w \in \Omega \qquad \Omega; \Gamma \vdash \Box A@w'}{\Omega; \Gamma \vdash \Box A@w} \qquad \texttt{(get)}\ \frac{w \in \Omega \qquad \Omega; \Gamma \vdash \Diamond A@w'}{\Omega; \Gamma \vdash \Diamond A@w}$$

Most of the rules here look rather natural. In particular `hyp`, $\to I$ and $\to E$ are standard and used also in $\mathcal{ND}$ systems for $\mathcal{IL}$. The new logical connectives rules $\Box E$ and $\Diamond I$ follow from the reflexivity axioms, $\Box I$ is motivated by the necessitation rule. Finally, $\Diamond E$ is a standard rule in any natural deduction system for intuitionistic modal logic.

Proofs of a number of axioms of IS5 are given below. We begin with a proof for distribution axiom $\Box(A \to B) \to (\Box A \to \Box B)$. Let w be any world from Ω and let $\Gamma = [\Box A@w, \Box(A \to B)@w]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Box(A \to B)@w \in \Gamma}{w_0 : \Omega; \Gamma \vdash \Box(A \to B)@w}\, w_0 \in w_0 : \Omega}{w_0 : \Omega; \Gamma \vdash (A \to B)@w_0} \qquad \dfrac{\dfrac{\Box A@w \in \Gamma}{w_0 : \Omega; \Gamma \vdash \Box A@w}\, w_0 \in w_0 : \Omega}{w_0 : \Omega; \Gamma \vdash A@w_0}}{w_0 : \Omega; \Gamma \vdash B@w_0}}{\Omega; \Gamma \vdash \Box B@w}\, w \in \Omega, \text{fresh } w_0}{\Omega; [\Box(A \to B)@w] \vdash (\Box A \to \Box B)@w}}{\Omega; \emptyset \vdash \Box(A \to B) \to (\Box A \to \Box B)@w}$$

Next example is $(\Diamond A \to \Box B) \to \Box(A \to B)$. Again, w can be any world from Ω and by Γ we mean $[A@w_0, \Diamond A \to \Box B@w]$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Diamond A \to \Box B@w \in \Gamma}{w_0 : \Omega; \Gamma \vdash \Diamond A \to \Box B@w} \qquad \dfrac{\dfrac{\dfrac{A@w_0 \in \Gamma}{w_0 : \Omega; \Gamma \vdash A@w_0}}{w_0 : \Omega; \Gamma \vdash \Diamond A@w_0}}{w_0 : \Omega; \Gamma \vdash \Diamond A@w}\, w_0 \in w_0 : \Omega}{w_0 : \Omega; \Gamma \vdash \Box B@w}}{w_0 : \Omega; \Gamma \vdash B@w_0}\, w_0 \in w_0 : \Omega}{w_0 : \Omega; [\Diamond A \to \Box B@w] \vdash (A \to B)@w_0}}{\Omega; [\Diamond A \to \Box B@w] \vdash \Box(A \to B)@w}\, w \in \Omega, \text{fresh } w_0}{\Omega; \emptyset \vdash (\Diamond A \to \Box B) \to \Box(A \to B)@w}$$

Reflexivity axioms for both $\Box$ and $\Diamond$ is actually captured in the rules, so we will skip those.

Next, transitivity in the $\Diamond$ case: $\Diamond\Diamond A \to \Diamond A$ in $w \in \Omega$.

$$\dfrac{\dfrac{\Diamond\Diamond A@w \in [\Diamond\Diamond A@w]}{\Omega; [\Diamond\Diamond A@w] \vdash \Diamond\Diamond A@w} \quad \dfrac{\dfrac{\Diamond A@w_0 \in [\Diamond A@w_0, \Diamond\Diamond A@w]}{w_0 : \Omega; [\Diamond A@w_0, \Diamond\Diamond A@w] \vdash \Diamond A@w_0}}{w_0 : \Omega; [\Diamond A@w_0, \Diamond\Diamond A@w] \vdash \Diamond A@w}\, w \in \Omega}{\dfrac{\Omega; [\Diamond\Diamond A@w] \vdash \Diamond A@w}{\Omega; \emptyset \vdash (\Diamond\Diamond A \to \Diamond A)@w}}$$

We will conclude with the proof of axiom **S5**: $\Diamond A \to \Box\Diamond A$ in any $w \in \Omega$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\Diamond A@w \in [\Diamond A@w]}{w_0 : \Omega; [\Diamond A@w] \vdash \Diamond A@w}}{w_0 : \Omega; [\Diamond A@w] \vdash \Diamond A@w_0}\, w_0 \in w_0 : \Omega}{\Omega; [\Diamond A@w] \vdash \Box\Diamond A@w}\, \text{fresh } w_0, w \in \Omega}{\Omega; \emptyset \vdash (\Diamond A \to \Box\Diamond A)@w}$$

Soundness and completeness

Again, we want to be sure that the system is sound and complete. Proofs for the two theorems below can be found in Simpson's PhD thesis [18].

Theorem 2.3 (Soundness)**.** Let Ω be a set of known worlds. For every theorem ϕ of IS5$_{\mathrm{L}}$: $\Omega; \emptyset \vdash \phi$, it follows that $\models \phi$.

Proof. Induction on structure of derivation $\Omega; \emptyset \vdash \phi$.

Theorem 2.4 (Completeness)**.** Let Ω be a set of known worlds. If $\models \phi$ then $\Omega; \emptyset \vdash \phi$.

Alternative ND systems - IS5$_{\mathbf{LF}}$ and IS5$_{\mathbf{Hyb}}$

Before we move towards applications, we would like to introduce two alternative natural deduction-style formalizations of IS5.
The first is due to a paper by Galmiche and Salhi [7]. It does not use world names at all, instead using a separate context for each world. We will refer to it as IS5$_{\mathrm{LF}}$.

The syntax used in IS5$_{\mathrm{LF}}$'s $\mathcal{ND}$ system uses the following:

Γ is the current context, about which we reason

G is what we call a background – it is simply an environment containing all the other, non-current contexts

IS5$_{\mathbf{LF}}$

$$\texttt{(hyp)}\ \frac{A \in \Gamma}{G \vdash \Gamma \vdash A}$$

$$(\to I)\ \frac{G \vdash A\colon \Gamma \vdash B}{G \vdash \Gamma \vdash A \to B} \qquad (\to E)\ \frac{G \vdash \Gamma \vdash A \to B \qquad G \vdash \Gamma \vdash A}{G \vdash \Gamma \vdash B}$$

$$(\Box I)\ \frac{\Gamma\colon G \vdash \emptyset \vdash A}{G \vdash \Gamma \vdash \Box A} \qquad (\Box E_1)\ \frac{G \vdash \Gamma \vdash \Box A}{G \vdash \Gamma \vdash A} \qquad (\Box E_2)\ \frac{\Gamma'\colon G \vdash \Gamma \vdash \Box A}{\Gamma\colon G \vdash \Gamma' \vdash A}$$

$$(\Diamond I_1)\ \frac{G \vdash \Gamma \vdash A}{G \vdash \Gamma \vdash \Diamond A} \qquad (\Diamond I_2)\ \frac{\Gamma'\colon G \vdash \Gamma \vdash A}{\Gamma\colon G \vdash \Gamma' \vdash \Diamond A}$$

$$(\Diamond E_1)\ \frac{G \vdash \Gamma \vdash \Diamond A \qquad (A\colon \emptyset)\colon G \vdash \Gamma \vdash B}{G \vdash \Gamma \vdash B}$$

$$(\Diamond E_2)\ \frac{\Gamma\colon G \vdash \Gamma' \vdash \Diamond A \qquad (A\colon \emptyset)\colon \Gamma'\colon G \vdash \Gamma \vdash B}{\Gamma'\colon G \vdash \Gamma \vdash B}$$

Note that for modal operators' introduction and elimination rules, we usually have two variants: one changing the current context to something from the background and the other leaving it untouched – this allows skipping get and fetch rules. Other than that, the rules correspond to ones from $\mathrm{L}_{\text{IS5-L}}$.

The second logic is our contribution - a variant of $\mathrm{IS5}_{\mathrm{LF}}$, but with explicit world names as in $\mathrm{IS5}_{\mathrm{L}}$. We will call it $\mathrm{IS5}_{\mathrm{Hyb}}$ as it is a hybrid between the other two and it makes a stepping stone in connecting (formally) the two. The syntax is the same as in $\mathcal{ND}$ for $\mathrm{IS5}_{\mathrm{LF}}$, except we name the contexts.

IS5$_{\mathbf{Hyb}}$

$$\texttt{(hyp)}\ \frac{A \in \Gamma}{G \vdash (w, \Gamma) \vdash A}$$

$$(\to I)\ \frac{G \vdash (w, A\colon \Gamma) \vdash B}{G \vdash (w, \Gamma) \vdash A \to B} \qquad (\to E)\ \frac{G \vdash (w, \Gamma) \vdash A \to B \qquad G \vdash (w, \Gamma) \vdash A}{G \vdash (w, \Gamma) \vdash B}$$

$$(\Box I)\ \frac{\text{fresh } w_0 \qquad (w, \Gamma)\colon G \vdash (w_0, \emptyset) \vdash A}{G \vdash (w, \Gamma) \vdash \Box A}$$

$$(\Box E_1)\ \frac{G \vdash (w, \Gamma) \vdash \Box A}{G \vdash (w, \Gamma) \vdash A} \qquad (\Box E_2)\ \frac{(w', \Gamma')\colon G \vdash (w, \Gamma) \vdash \Box A}{(w, \Gamma)\colon G \vdash (w', \Gamma') \vdash A}$$

$$(\Diamond I_1)\ \frac{G \vdash (w,\Gamma) \vdash A}{G \vdash (w,\Gamma) \vdash \Diamond A} \qquad (\Diamond I_2)\ \frac{(w',\Gamma') : G \vdash (w,\Gamma) \vdash A}{(w,\Gamma) : G \vdash (w',\Gamma') \vdash \Diamond A}$$

$$(\Diamond E_1)\ \frac{G \vdash (w,\Gamma) \vdash \Diamond A \quad \text{fresh } w_0 \quad ((w_0,A) : \emptyset) : G \vdash (w,\Gamma) \vdash B}{G \vdash (w,\Gamma) \vdash B}$$

$$(\Diamond E_2)\ \frac{(w',\Gamma') : G \vdash (w,\Gamma) \vdash \Diamond A \quad \text{fresh } w_0 \quad (w_0, A : \emptyset) : (w,\Gamma) : G \vdash (w',\Gamma') \vdash B}{(w,\Gamma) : G \vdash (w',\Gamma') \vdash B}$$

2.4 Applications

What are possible applications of intuitionistic modal logics? They vary from programming languages to describing behavior of hardware circuits, from applications in bisimularity to staged computations. We would now like to focus on applications for IS5, namely distributed programming, as it is our main motivation for investigating IS5.

How do we define a problem for which, as we claim, IS5 is the solution? Imagine we have a number of hosts in the network. The connectivity relation R will determine how these hosts are connected. In a standard networking environment it is most natural to assume that all hosts are connected to one another, so R is an equivalence. This is exactly the case in IS5.
Next, some resources are available only locally. It means that each host in the network has direct access to its own resources, but not to the resources of its neighbors. This characterizes type $\Diamond$: it is a type for local address. We can have such address, but unless we are at the host, at which this address was made, we cannot unpack it and use its value.
Finally, for some programs to be run in a distributed environment, they need to be executable from any host within the network – we will call them *mobile*. We want to be able to express that for some computations it does not matter where they are being run. This matches our understanding of necessity ("true everywhere") – making $\Box$ type a good choice for such mobile programs.

With such interpretation, what do we make of some of the the characteristic propositions from IS5?

$\Box A \to A$ is simple to interpret: mobile programs are executable

$A \to \Diamond A$ is just as straightforward, stating that we can create an address of anything

$\Diamond\Box A \to \Box A$ states that if we have an address of a mobile value, then we can obtain such value

Having said all that, it suggests that IS5 seems like a good type system for distributed programming languages. What is missing are actual terms. We will add them in the next chapter, but first let us talk a little bit about how we can actually put the logic into a good use as type system via Curry-Howard correspondence.

2.4.1 Curry-Howard isomorphism

Originally discovered by Haskell Curry and William Alvin Howard, by Curry-Howard isomophism we now understand a relationship between proofs in a logical system and programs. To put things simply: *a proof is a program, the formula it proves is a type for this program.* Furthermore, what this means is that reduction (when looking at terms) can be viewed as proof transformation for logic.

The simplest but perhaps most vivid example is the original observation of Howard, correlation between the intuitionistic natural deduction system and the simply typed λ-calculus. We will now give a brief overview of the two. For the sake of simplicity we will limit ourselves to a system with only $\to$ operator.

Natural deduction

Natural deduction system for implication has the following rules of inference, where α and β are judgments "α is true" and "β is true" respectively.

$$(\text{Ax})\ \frac{\alpha \in \Gamma}{\Gamma \vdash \alpha} \qquad (\to I)\ \frac{\alpha\colon \Gamma \vdash \beta}{\Gamma \vdash \alpha \to \beta} \qquad (\to E)\ \frac{\Gamma \vdash \alpha \to \beta \qquad \Gamma \vdash \alpha}{\Gamma \vdash \beta}$$

For example, a proof of the judgment $(\alpha \to \beta) \to \alpha \to \beta$ being true can be constructed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\alpha \to \beta) \in [\alpha, \alpha \to \beta]}{[\alpha, \alpha \to \beta] \vdash \alpha \to \beta} \qquad \dfrac{\alpha \in [\alpha, \alpha \to \beta]}{[\alpha, \alpha \to \beta] \vdash \alpha}}{[\alpha, \alpha \to \beta] \vdash \beta}}{[\alpha \to \beta] \vdash \alpha \to \beta}}{[\,] \vdash (\alpha \to \beta) \to \alpha \to \beta}$$

Note that there may be more than one proof tree for a given statement. Take, for example, $\alpha \to \alpha$. A proof for that can simply be

$$\dfrac{\dfrac{\alpha \in [\alpha]}{[\alpha] \vdash \alpha}}{[\,] \vdash \alpha \to \alpha}$$

But nothing stops us from producing the following proof:

$$\dfrac{\dfrac{(\alpha \to \alpha) \in [\alpha \to \alpha]}{[\,] \vdash (\alpha \to \alpha) \to (\alpha \to \alpha)} \quad \dfrac{\dfrac{\alpha \in [\alpha]}{[\alpha] \vdash \alpha}}{[\,] \vdash \alpha \to \alpha}}{[\,] \vdash \alpha \to \alpha}$$

λ-calculus with simple types

Typing rules for simple λ-calculus are the following:

$$\text{(Var)}\ \frac{x\colon \alpha \in \Gamma}{\Gamma \vdash x\colon \alpha}$$

$$\text{(Lam)}\ \frac{(x\colon \alpha)\colon \Gamma \vdash M\colon \beta}{\Gamma \vdash \lambda x^{\alpha}.M\colon \alpha \to \beta} \qquad \text{(Appl)}\ \frac{\Gamma \vdash M\colon \alpha \to \beta \qquad \Gamma \vdash N\colon \alpha}{\Gamma \vdash M \cdot N\colon \beta}$$

Note that α and β are now types for terms in λ-calculus.

Looking at the rules, the similarity is striking. Let us then take a look at two different terms: first is an identity function $\lambda x^{\alpha}.x$, the second is more complex: $(\lambda f^{(\alpha\to\alpha)}.f) \cdot (\lambda x^{\alpha}.x)$. We want to assign them types using typing rules we have just provided.

First the identity function:

$$\dfrac{\dfrac{x\colon \alpha \in [x\colon \alpha]}{[x\colon \alpha] \vdash x\colon \alpha}}{[\,] \vdash \lambda x^{\alpha}.x\colon \alpha \to \alpha}$$

And then the more complex term:

$$\dfrac{\dfrac{(f\colon \alpha \to \alpha) \in [f\colon \alpha \to \alpha]}{[\,] \vdash \lambda f^{(\alpha\to\alpha)}.f(\alpha \to \alpha) \to (\alpha \to \alpha)} \quad \dfrac{\dfrac{x\colon \alpha \in [x\colon \alpha]}{[x\colon \alpha] \vdash x\colon \alpha}}{[\,] \vdash \lambda x^{\alpha}.x\colon \alpha \to \alpha}}{[\,] \vdash (\lambda f^{(\alpha\to\alpha)}.f) \cdot (\lambda x^{\alpha}.x)\colon \alpha \to \alpha}$$

Not only they are of the same type, their typing trees look exactly like two proofs for $\alpha \to \alpha$ in natural deduction system. When checking the type of term $\lambda x^{\alpha}.x$ we did not have any choice – we had to build the simpler proof tree. This is true in general and this is what we mean by "programs encode proofs".

Computations

For λ-calculus the computation consists of applying β-reduction until a value is obtained: $(\lambda x^{\alpha}.M) \cdot N \mapsto_{\beta} [N|x]M$. Without changing the final result of the computation, we can also expand the term via η-expansion: $M \mapsto_{\eta} \lambda x^{\alpha}.(M \cdot x)$ when $x \notin \mathrm{FV}\, M$. What is the meaning of these operations in logic?

β-reduction corresponds to local soundness of $\to$ in natural deduction, a form of optimizing the proof. Looking at the proof tree:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\(x\colon \alpha)\colon \Gamma \vdash M\colon \beta\end{array}}{\Gamma \vdash \lambda x^{\alpha}.M\colon \alpha \to \beta} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma \vdash N\colon \alpha\end{array}}{\Gamma \vdash (\lambda x^{\alpha}.M) \cdot N\colon \beta}$$

we can see that it uses $\to$ introduction one step before elimination. In pure logic without terms, we can rewrite this proof as

$$\dfrac{[\mathcal{E}|\alpha]\mathcal{D}}{\Gamma \vdash N\colon \beta}$$

Similarly, η-expansion is justified by local completeness of $\to$ operator.

We have advocated the importance of finding an interpretation for some logic – it is not because we can formally describe e.g. distributed computations using IS5, but because we can obtain a programming language that realizes this interpretation. In the next chapter we will present three such languages, one for each variant of IS5 natural deduction system.

Chapter 3

Term languages for IS5

In this chapter we describe syntax and operational semantics for languages corresponding to three variants of IS5's $\mathcal{ND}$ systems presented in the previous chapter. We also justify correctness of one of these languages by presenting proofs of progress and preservation.

3.1 Syntax

Keeping in mind applications mentioned in the previous chapter, we begin by describing the syntax of a prototypical programming language[1] for distributed computations. This language will use IS5 as a type system via the Curry-Howard isomorphism. This will allow us to relate proofs in IS5 with programs in $\mathrm{L}_{\mathrm{IS5}}$, as mentioned previously.

Note that as our interest lies in the properties of the formal system rather than the practicality of the language, our prototype is a λ-calculus-like language rather then, say, SML-like one. In particular, it is not intended for actual programming, but it could be enriched in various constructs like ML5, a real-life language with IS5-based type system([14], [13], [12]).

3.1.1 Non-modal terms

The specific language features we are interested concern mobility, therefore for the non-modal terms we limit ourselves to functions and variable usage – adding pairs as interpretation for conjunction should not cause problems,

[1] When giving examples of inference rules throughout this section, we will use $\mathrm{L}_{\mathrm{IS5\text{-}L}}$, which will be formally introduced later in this chapter.

other typing rules may prove more challenging – [7] is a good starting point for adding them.

Our syntax for this part is identical to the standard lambda calculus:

$\lambda(v : A).M$ is a unary function term, taking one argument of type A and computing term M. Term M may contain free occurrences of variable v, which is bound under λ.

$M \cdot N$ is an application of term M to N.

$\texttt{hyp}\, v$ marks assumption of a certain variable v.

Typing rules for $\lambda(v : A).M$ and $M \cdot N$ correspond to $\rightarrow$ introduction and elimination rules in IS5.

3.1.2 Mobile code

The rest of the syntax for the language of distributed computations is new – we will start with type $\Box A$ for *mobile* code of type A. When we say 'mobile code' we mean code that can be executed in any location in the network, regardless of its place of origin.

Since we want to be able to run mobile code in every accessible world, we need means to move between them. This is achieved using $\texttt{fetch}\ w\ M$, an operation explicitly moving execution from one place to another. We only want to allow mobile code to be moved in this manner. The rule `(fetch)` below captures this restriction in a type system for $\mathrm{L}_{\text{IS5-L}}$, one of the languages we will describe later in this chapter:

$$\texttt{(fetch)}\ \frac{w' \in \Omega \qquad \Omega; \Gamma \vdash M : \Box A @ w}{\Omega; \Gamma \vdash \texttt{fetch}\ w\ M : \Box A @ w'}$$

Note that the only requirement we have apart from M being of mobile type is that host w' is known to us.

We know how to move the mobile code from one world to another. What about declaring something to be mobile? This is done using $\texttt{box}\ w\ M$.
As we have already mentioned, mobile code can be executed anywhere. Treating this description as an actual specification, we may say that for code M to be mobile, we require it to have a certain type A in every accessible world. There is however a problem with such an approach.

What is the status of shared memory? Do we have it? Do we want it? Say that we have assumption $v : A$ in each of known worlds. Does this mean that such assumption $\texttt{hyp}\, v$ should be mobile? Probably not. One good reason is that we want to be able to expand the set of known worlds, possibly with a world that has no assumptions in its context. After such expansion (usually referred to as weakening) our once mobile code $\texttt{hyp}\, v$ stops being mobile. But how do we prevent this from happening? The simplest way is to require for M to have type A in a fresh world, of which nothing is yet known. If we can construct a type without any assumptions, we can also do it with them. The appropriate rule in $\mathrm{L}_{\mathrm{IS5\text{-}L}}$ is of the form:

$$\texttt{(box)}\ \frac{\text{fresh}\ w_0 \qquad w \in \Omega \qquad w_0 : \Omega; \Gamma \vdash M : A@w_0}{\Omega; \Gamma \vdash \texttt{box}\ \ M : \Box A@w}$$

Freshness of w_0 in this case means that Γ does not contain any assumptions of the form $x : A@w_0$ and that w_0 is not known in Ω.

Lastly, to actually execute code that is considered to be mobile, we need to be able to remove the mobility marker $\Box$. This is done using the unbox operator and corresponds to $\Box$ elimination rule. Of course, if it happens that introduction and elimination rules are used one after the other when typing a term $(\texttt{unbox}\ \ (\texttt{box}\ \ w_0\ M))$, we can simply use construction used to type M in a fresh world w_0. It will surely be a correct proof tree as it uses no assumptions about the current world.

3.1.3 Addresses

Dually to universally executable modal code, we can introduce remote addresses. If code M has type A at world w, then in this very world we can obtain the address of such code, which will be a term $\texttt{here}\ \ M$ of type $\Diamond A$. The modality $\Diamond$ denotes address.

In order to move a term of address type between hosts, we will need a get operator, strongly resembling fetch, but used on addresses rather than mobile code.

Finally, how do we use addresses? Suppose that in order to type a term N we require some knowledge about code of type A. We do not know what this code looks like or where it can be found, only that it is somewhere within the network. We can emulate such knowledge by adding a variable v of type A to a freshly created host w_0.

Now, once we have found some term M that is of type $\Diamond A$, we can remove the fake fresh host. The operator to do so is called letdia.
An example of its formalization in $\mathrm{L}_{\text{IS5-L}}$ would look like this:

$$\texttt{(letdia)}\ \frac{\Omega;\Gamma \vdash M : \Diamond A@w \qquad \text{fresh } w_0 \qquad w_0 : \Omega; (v : A@w_0) : \Gamma \vdash N : B@w}{\Omega;\Gamma \vdash \texttt{letdia } [v_{w_0} := M]\ N : B@w}$$

Note that we do not yet make any real substitutions - term N will still contain occurrences of variable v. Instead, we have knowledge of an address of A-typed term, thus ensuring that it will eventually be able to remove the variable v and use a real value instead. If, however, we happen to be in a situation where $M = \texttt{here } M_0$, then intuitively we can replace v with M_0, as it is of the right type – A.

3.2 A labeled language

The first formalization we will look into in more details comes from [14]. It focuses on the global state of a distributed system, as we only have one context containing all the assumptions.

3.2.1 Type system

We will start by giving typing rules for the system. Some of them are standard and identical as in natural deduction systems for intuitionistic logic (except using slightly different judgments), others come from new operators, i.e.: `box`, `unbox`, `fetch`, `here`, `letdia` and `get`. The previous section provides intent for their interpretation.

Let us review the notations used in $\mathrm{L}_{\text{IS5-L}}$:

Ω denotes a set of known worlds – hosts in the network

Γ is a global context, containing assumptions of the form $v : A@w$. We require each assumption's name v to be unique - not only within its host w, but globally.

$A@w$ reads "type A on host w"

Syntax for terms of $\mathrm{L}_{\text{IS5-L}}$ is summarized as:

$$M := \texttt{hyp}\, v \mid \lambda(v : \tau).M \mid M \cdot M \mid \texttt{box } w\ M \mid \texttt{unbox } M \mid \texttt{fetch } w\ M \mid \texttt{here } M \mid \texttt{letdia } [v := w]\ MM \mid \texttt{get } w\ M$$

where w denotes worlds, v denotes variables and τ denotes types (with ι denoting the base type): $\tau := \iota \mid \tau \to \tau \mid \Box\tau \mid \Diamond\tau$.

$$\texttt{(hyp)}\ \frac{w \in \Omega \qquad (v\colon A@w) \in \Gamma}{\Omega;\Gamma \vdash \texttt{hyp}\, v\colon A@w}$$

$$\texttt{(lambda)}\ \frac{\text{fresh } v_0 \qquad \Omega;(v_0\colon A@w)\colon \Gamma \vdash M\colon B@w}{\Omega;\Gamma \vdash \lambda(v_0 : A).M\colon (A \to B)@w}$$

$$\texttt{(appl)}\ \frac{\Omega;\Gamma \vdash M\colon (A \to B)@w \qquad \Omega;\Gamma \vdash N\colon A@w}{\Omega;\Gamma \vdash M \cdot N\colon B@w}$$

$$\texttt{(box)}\ \frac{w \in \Omega \qquad \text{fresh } w_0 \qquad w_0\colon \Omega;\Gamma \vdash M\colon A@w_0}{\Omega;\Gamma \vdash \texttt{box}\ \ w_0\ M\colon \Box A@w}$$

$$\texttt{(fetch)}\ \frac{w \in \Omega \qquad \Omega;\Gamma \vdash M\colon \Box A@w'}{\Omega;\Gamma \vdash \texttt{fetch}\ \ w'\ M\colon \Box A@w} \qquad \texttt{(unbox)}\ \frac{\Omega;\Gamma \vdash M\colon \Box A@w}{\Omega;\Gamma \vdash \texttt{unbox}\ \ M\colon A@w}$$

$$\texttt{(here)}\ \frac{\Omega;\Gamma \vdash M\colon A@w}{\Omega;\Gamma \vdash \texttt{here}\ \ M\colon \Diamond A@w} \qquad \texttt{(get)}\ \frac{w \in \Omega \qquad \Omega;\Gamma \vdash M\colon \Diamond A@w'}{\Omega;\Gamma \vdash \texttt{get}\ \ w'\ M\colon \Diamond A@w}$$

$$\texttt{(letdia)}\ \frac{\Omega;\Gamma \vdash M\colon \Diamond A@w\ \ \text{fresh } w_0, \text{fresh } v\ \ w_0\colon \Omega;(v\colon A@w_0)\colon \Gamma \vdash N\colon B@w}{\Omega;\Gamma \vdash \texttt{letdia}\ \ [v_{w_0} := M]\ N\colon B@w}$$

3.2.2 Operational semantics

We are now able to write type-correct programs in $\mathrm{L}_{\mathrm{IS5\text{-}L}}$; the next step is to assign them meaning by giving structural operational semantics to each term. There are of course expressions we consider to be final – values. First, a function is a value as usual, since we do not know what is the given argument and therefore cannot continue evaluation. Next, $\texttt{box}\ \ w_0\ M$ is a value, as for the mobile code we do not know what host we will run it at. Finally, for an address $\texttt{get}\ \ w\ (\texttt{here}\ \ M)$ to be a value we require that the term M is a value as well. We will denote the fact that M is a value by $\mathrm{val}(M)$ further on.

For some of the reductions we are about to describe we will require operations we haven't mentioned before – term substitution $[M|v]N$ and world merge $\{w|w'\}N$. They are both defined inductively on N, their interpretations are standard: $[M|v]N$ replaces each occurrence of $\texttt{hyp}\, v$ by M and $\{w|w'\}N$ replaces each occurrence of world w' by world w, effectively merging these two into one.

Final remark is that reduction in $\mathrm{L}_{\mathrm{IS5\text{-}L}}$ is a relation not just between terms. It requires also knowledge about the current host. To see that, con-

sider what should happen when we are evaluating a program of the form $\texttt{unbox}\ (\texttt{box}\ w_0\ M)$. We want to use (that is, unbox) mobile code we have just created ($\texttt{box}\ w_0\ M$), but to do so, we need to know at what host we will evaluate M. We will denote reduction taking place at host w as $\mapsto_w$.

The most interesting part of operational semantics comes from local soundness of the connectives. We begin with necessity introduction followed by its elimination case that we have used to motivate introduction of host parameter into reduction relation:

$$\texttt{unbox}\ (\texttt{box}\ w_0\ M) \mapsto_w \{w|w_0\}M$$

Intuitively, this is clear – we want to execute code that has just been made mobile, so we are simply replacing fresh name w_0 with the actual one – w. This can be justified by the following reduction (simplification) of the proof tree:

$$\Box E\,\dfrac{\Box I\,\dfrac{\begin{array}{c}\mathcal{D}\\ w_0\colon \Omega;\Gamma \vdash M\colon A@w_0\end{array}}{\Omega;\Gamma \vdash \texttt{box}\ w_0\ M\colon \Box A@w}\ w \in \Omega, \text{fresh}\ w_0}{\Omega;\Gamma \vdash \texttt{unbox}\ (\texttt{box}\ w_0\ M)\colon A@w} \quad \Rightarrow_R \quad \begin{array}{c}\{w|w_0\}\mathcal{D}\\ \Omega;\Gamma \vdash \{w|w_0\}M\colon A@w\end{array}$$

When writing $\{w|w_0\}\mathcal{D}$ we mean merging w with w_0 through all: $w_0\colon \Omega$, Γ, the term, and the type part.
In particular for the last judgment, we have an assumption $w \in \Omega$ and we treat Ω as a set. In addition $\{w|w_0\}\Gamma = \Gamma$, since we chose w_0 to be fresh - thus not known in Γ. Therefore:
$\{w|w_0\}(w_0\colon \Omega;\Gamma \vdash M\colon A@w_0) = \Omega;\Gamma \vdash \{w|w_0\}M\colon A@w$.

Next, let us take a closer look at local soundness of addresses:

$$\texttt{letdia}\ [v_{w_0} := \texttt{get}\ w'\ (\texttt{here}\ M)]\ N \mapsto_w [M|v]\{w'|w_0\}N$$

Again, the proof tree justifies the reduction (w_0 and v are fresh outside $\mathcal{E}$ subtree):

$$\Diamond E\,\dfrac{\Diamond I\,\dfrac{\begin{array}{c}\mathcal{D}\\ \Omega;\Gamma \vdash M\colon A@w'\end{array}}{\Omega;\Gamma \vdash \texttt{here}\ M\colon \Diamond A@w'}}{\Omega;\Gamma \vdash \texttt{get}\ w'\ (\texttt{here}\ M)\colon \Diamond A@w} \qquad \begin{array}{c}\mathcal{E}\\ w_0\colon \Omega;(v\colon A@w_0)\colon \Gamma \vdash N\colon B@w\end{array}}{\Omega;\Gamma \vdash \texttt{letdia}\ [v_{w_0} := \texttt{get}\ w'\ (\texttt{here}\ M)]\ N\colon B@w}$$

$$\Rightarrow_R \quad \begin{array}{c}[\mathcal{D}|v]\{w'|w_0\}\mathcal{E}\\ \Omega;\Gamma \vdash [M|v]\{w'|w_0\}N\colon B@w\end{array}$$

We have discussed most substitutions used here in the previous case, the only new one is $[\mathcal{D}|v]\{w'|w_0\}\mathcal{E}$ – this just means using proof tree $\mathcal{D}$ whenever we want to use the fact that v is a variable of type A in a given context.

Next, there are compatibility rules for the new connectives:

if $M \mapsto_w M'$, then $\texttt{unbox}\ \ M \mapsto_w \texttt{unbox}\ \ M'$

if $M \mapsto_{w'} M'$, then $\texttt{fetch}\ \ w'\ M \mapsto_w \texttt{fetch}\ \ w'\ M'$

if $M \mapsto_{w'} M'$, then $\texttt{get}\ \ w'\ M \mapsto_w \texttt{get}\ \ w'\ M'$

if $M \mapsto_w M'$, then $\texttt{letdia}\ \ [v_{w_0} := M]\ N \mapsto_w \texttt{letdia}\ \ [v_{w_0} := M']\ N$

if $M \mapsto_w M'$, then $\texttt{here}\ \ M \mapsto_w \texttt{here}\ \ M'$

along with two special rules for moving between hosts, that do require a comment:

if $\text{val}(M)$, then $\texttt{fetch}\ \ w'\ M \mapsto_w M$

if $\text{val}(M)$, then $\texttt{here}\ \ M \mapsto_w \texttt{get}\ \ w\ (\texttt{here}\ \ M)$

The rule for `here` operator is motivated by looking at definitions of values in $\text{L}_{\text{IS5-L}}$. $\texttt{get}\ \ w\ (\texttt{here}\ \ M)$ will already be a value since M is a value – thus, the purpose of this rule is to mark the end of evaluation. Instead of this rule we can add "if $\text{val}(M)$, then $\text{val}(\texttt{here}\ \ M)$" to the set of values.
The first rule also seems artificial, but note that in the `fetch` case for any well-typed program, M must be of the form $\texttt{box}\ \ w_0\ M'$ (since it is a value) - we can therefore safely skip any world merging and just choose a different host when declaring code to be mobile.

Finally we have β-reduction and the rules expressing compatibility with respect to application:

β-reduction:	$(\lambda(v : A).M) \cdot N \mapsto_w [N|v]M$
compatibility with application:	if $M \mapsto_w M'$, then $M \cdot N \mapsto_w M' \cdot N$

3.3 A label-free language

Looking at the global state of the system, as in the previous section, is not the only possibility for environment formalization. The label-free logic IS5_{LF} described in [7] uses a multi-contextual environment where each host has its

own context. There is however no language given as an application for $\mathrm{IS5_{LF}}$. We fill that gap here by introducing $\mathrm{L_{IS5\text{-}LF}}$.

In this particular formalization we do not give names to the hosts at all. There is no way to distinguish between two hosts containing the same assumptions (or no assumptions at all). However, as we do require all variable names to be distinctive, only hosts with no assumptions are truly indistinguishable one from another.

This changes interpretation a little bit, as we cannot use host names in our programs. Therefore we have no way of delegating execution to some specific host. One might think that it should be possible to use sets of assumptions true in specific hosts instead of host names, but this does not seem feasible, as these assumptions may change during execution when contexts are expanded (e.g. rule for $\lambda(v : A).M$). As a result, having only code of a program in $\mathrm{L_{IS5\text{-}LF}}$ does not give us enough information to tell if this program uses resources outside its host. Only a proof tree can show that information.

We will split all contexts into the current context Γ and the background G. By that we mean that computations take place in Γ and all hosts which are connected to the current one are in G.

3.3.1 Type system

As there are no rules like `(fetch)` or `(get)` from $\mathrm{L_{IS5\text{-}L}}$, we can only move between hosts upon creating or using modal-typed terms. This is why there are two ways of creating terms `here`, `unbox` and `letdia`.
In the article[7] that introduces this variant of IS5 logic the authors did not add terms at all. They did, however, use two variants of $\Box E$, $\Diamond I$ and $\Diamond E$ rules, one changing the current context into another one from the background and the other leaving it with no change. Our decision not to differentiate these two situations also at the term level was a technical one. Granted, using different terms (say, `unbox` and `unbox-fetch`) in these two situations would make it possible to tell – just from the program – if it is run locally or if it needs external resources. But there were too many complications arising from such decision; for example, when we are merging two hosts' contexts, how do we tell if some `unbox-fetch` instance should not be turned into `unbox` after the merge? This would require looking at the entire proof tree! Therefore, we have only one term for both variants of `(unbox)` rule (same for `(here)` and `(letdia)`).

Syntax for terms of $\mathrm{L}_{\mathrm{IS5\text{-}LF}}$ is summarized as:

$$\begin{aligned} M \;:=\; & \mathtt{hyp}\, v \mid \lambda(v:\tau).M \mid M \cdot M \mid \mathtt{box}\; M \mid \mathtt{unbox}\;\; M \\ & \mid \mathtt{here}\;\; M \mid \mathtt{letdia}\;\; [v := M]\; M \end{aligned}$$

$$\mathtt{(hyp)}\;\frac{(v\colon A) \in \Gamma}{G \vdash \Gamma \vdash \mathtt{hyp}\, v\colon A}$$

$$\mathtt{(lambda)}\;\frac{\text{fresh}\;\; v_0 \qquad G \vdash (v_0\colon A)\colon \Gamma \vdash M\colon B}{G \vdash \Gamma \vdash \lambda(v_0 : A).M\colon A \to B}$$

$$\mathtt{(appl)}\;\frac{G \vdash \Gamma \vdash M\colon A \to B \qquad G \vdash \Gamma \vdash N\colon A}{G \vdash \Gamma \vdash M \cdot N\colon B}$$

$$\mathtt{(box)}\;\frac{\Gamma\colon G \vdash \emptyset \vdash M\colon A}{G \vdash \Gamma \vdash \mathtt{box}\; M\colon \Box A}$$

$$\mathtt{(unbox)}\;\frac{G \vdash \Gamma \vdash M\colon \Box A}{G \vdash \Gamma \vdash \mathtt{unbox}\;\; M\colon A} \qquad \mathtt{(unbox\text{-}fetch)}\;\frac{\Gamma'\colon G \vdash \Gamma \vdash M\colon \Box A}{\Gamma\colon G \vdash \Gamma' \vdash \mathtt{unbox}\;\; M\colon A}$$

$$\mathtt{(here)}\;\frac{G \vdash \Gamma \vdash M\colon A}{G \vdash \Gamma \vdash \mathtt{here}\;\; M\colon \Diamond A} \qquad \mathtt{(get\text{-}here)}\;\frac{\Gamma'\colon G \vdash \Gamma \vdash M\colon A}{\Gamma\colon G \vdash \Gamma' \vdash \mathtt{here}\;\; M\colon \Diamond A}$$

$$\mathtt{(letdia)}\;\frac{G \vdash \Gamma \vdash M\colon \Diamond A \qquad \text{fresh}\;\; v_0 \qquad [v_0\colon A]\colon G \vdash \Gamma \vdash N\colon B}{G \vdash \Gamma \vdash \mathtt{letdia}\;\; [v_0 := M]\; N\colon B}$$

$$\mathtt{(letdia\text{-}get)}\;\frac{\Gamma\colon G \vdash \Gamma' \vdash M\colon \Diamond A \qquad \text{fresh}\;\; v_0 \qquad [v_0\colon A]\colon \Gamma'\colon G \vdash \Gamma \vdash N\colon B}{\Gamma'\colon G \vdash \Gamma \vdash \mathtt{letdia}\;\; [v_0 := M]\; N\colon B}$$

As we can see, the rules of this language are a bit more natural; they do not use too many syntax extensions. Compare the rule for box operator in $\mathrm{L}_{\mathrm{IS5\text{-}LF}}$ with similar rule in $\mathrm{L}_{\mathrm{IS5\text{-}L}}$:

$$(\mathtt{box}_{\mathrm{LF}})\;\frac{\Gamma\colon G \vdash \emptyset \vdash M\colon A}{G \vdash \Gamma \vdash \mathtt{box}\; M\colon \Box A} \qquad (\mathtt{box}_{\mathrm{L}})\;\frac{w \in \Omega \qquad \text{fresh}\;\; w_0 \qquad w_0\colon \Omega; \Gamma \vdash M\colon A@w_0}{\Omega; \Gamma \vdash \mathtt{box}\;\; w_0\; M\colon \Box A@w}$$

We notice for example that to create a new world it is enough to just add an empty context to the environment. As there are no names, it must be fresh by definition.

Another fact to observe is the difference between `(here)` and `(get-here)` (or any other pair of constructors/destructors of the same type) and corresponding terms from $\mathrm{L}_{\mathrm{IS5\text{-}L}}$. In particular, `(get-here)` is exactly a result of flattening rules `(get)` and `(here)`:

$$\texttt{(get-here)}\ \frac{\Gamma' \colon G \vdash \Gamma \vdash M \colon A}{\Gamma \colon G \vdash \Gamma' \vdash \texttt{here}\ \ M \colon \Diamond A} \qquad \frac{w \in \Omega \qquad \dfrac{\Omega; \Gamma \vdash M \colon A@w'}{\Omega; \Gamma \vdash \texttt{here}\ \ M \colon \Diamond A@w'}\ \texttt{(here)}}{\Omega; \Gamma \vdash \texttt{get}\ \ w'\ (\texttt{here}\ \ M) \colon \Diamond A@w}\ \texttt{(get)}$$

3.3.2 Operational semantics

Operational semantics for $\mathrm{L}_{\mathrm{IS5\text{-}LF}}$ resembles the one for $\mathrm{L}_{\mathrm{IS5\text{-}L}}$. Values are the same: $\mathrm{val}(\lambda(v : A).M)$, $\mathrm{val}(\texttt{box}\ M)$, except there is no `get` operation, therefore `here` itself can be a value: if $\mathrm{val}(M)$, then $\mathrm{val}(\texttt{here}\ \ M)$.

Most reductions are also similar, but they do not use host name as an additional parameter.

As per its name, label-free formalization does not use labels for contexts at any point, so there is no need to define world merging operation on terms. Soundness-motivated reduction for $\Box$ type is therefore as simple as the following:

$$\texttt{unbox}\ \ (\texttt{box}\ M) \mapsto M$$

This is justified by the following proof trees – there are two as there are two possibilities of how `unbox` operation looks like – it either changes the host or it does not:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma \colon G \vdash \emptyset \vdash M \colon A}\,\Box I}{G \vdash \Gamma \vdash \texttt{box}\ M \colon \Box A}\,\Box E}{G \vdash \Gamma \vdash \texttt{unbox}\ \ (\texttt{box}\ M) \colon A} \quad \Rightarrow_R \quad \dfrac{\{\Gamma \cup \emptyset\}_{\mathrm{curr}}\mathcal{D}}{G \vdash \Gamma \vdash M \colon A}$$

$$\Box E\,\dfrac{\Box I\,\dfrac{\dfrac{\mathcal{D}}{\Gamma' \colon \Gamma \colon G \vdash \emptyset \vdash M \colon A}}{\Gamma \colon G \vdash \Gamma' \vdash \texttt{box}\ M \colon \Box A}}{\Gamma' \colon G \vdash \Gamma \vdash \texttt{unbox}\ \ (\texttt{box}\ M) \colon A} \quad \Rightarrow_R \quad \dfrac{\{\Gamma \cup \emptyset\}_{\mathrm{curr}}\mathcal{D}}{\Gamma' \colon G \vdash \Gamma \vdash M \colon A}$$

The operation $\{\Gamma \cup \Delta\}_{\mathrm{curr}}$ merges Γ and Δ when Δ is the current context. So in this case it merges Γ with $\emptyset$, effectively moving Γ to be the current context. In terms there is no distinction between moving between contexts and staying in place, so we do not have to change anything in M.
Note that $\{\Gamma \cup \emptyset\}_{\mathrm{curr}}(\Gamma \colon G \vdash \emptyset \vdash M \colon A) = G \vdash \Gamma \vdash M \colon A$ and

$\{\Gamma \cup \emptyset\}_{\text{curr}}(\Gamma' : \Gamma : G \vdash \emptyset \vdash M : A) = \Gamma' : G \vdash \Gamma \vdash M : A.$

Similarly for $\Diamond$ type we have:

$$\text{if } \text{val}(M), \text{ then } \texttt{letdia } [v := \texttt{here } M] \ N \mapsto [M|v]N$$

This is justified by:

$$\Diamond I \; \frac{\mathcal{D}}{\Diamond E \; \dfrac{G \vdash \Gamma \vdash M : A}{G \vdash \Gamma \vdash \texttt{here } M : \Diamond A} \qquad \text{fresh } v \qquad \dfrac{\mathcal{E}}{((v : A) : \emptyset) : G \vdash \Gamma \vdash N : B}}{G \vdash \Gamma \vdash \texttt{letdia } [v := \texttt{here } M] \ N : B@w}$$

$$\Longrightarrow_R \quad \frac{[\mathcal{D}|v]\{(v, A) : \emptyset \cup \Gamma\}_{\text{curr}} \ \mathcal{E}}{G \vdash \Gamma \vdash [M|v]N : B@w}$$

This is just one possible proof tree for $\texttt{letdia } [v := (\texttt{here } M)] \ N$, as both `letdia` and `here` can exchange current context, giving a total of four combinations.

The rest of the rules, including $\beta-$reduction and compatibility rules are as follows:

$$(\lambda(v : A).M) \cdot N \mapsto [N|v]M$$

$$\text{if } M \mapsto M', \text{ then } M \cdot N \mapsto M' \cdot N$$

$$\text{if } M \mapsto M', \text{ then } \texttt{unbox } M \mapsto \texttt{unbox } M'$$

$$\text{if } M \mapsto M', \text{ then } \texttt{here } M \mapsto \texttt{here } M'$$

$$\text{if } M \mapsto M', \text{ then } \texttt{letdia } [v := M] \ N \mapsto \texttt{letdia } [v := M'] \ N$$

3.4 A hybrid language

The third language we would like to introduce in this thesis, $\text{L}_{\text{IS5-Hyb}}$, is a combination of both previous ones. It uses multi-contextual environment, but at the same time still uses names for hosts. It is fairly simple to turn any program written in $\text{L}_{\text{IS5-Hyb}}$ into both $\text{L}_{\text{IS5-L}}$ and $\text{L}_{\text{IS5-LF}}$ – as we will see in the next chapter. In that sense it acts as a bridge between these two languages.

3.4.1 Type system

The type system for $\mathrm{L}_{\text{IS5-Hyb}}$ is derived from the one for $\mathrm{L}_{\text{IS5-LF}}$, except each context has a name. We require that variable names be unique, as well as host names.

Syntax for terms of $\mathrm{L}_{\text{IS5-Hyb}}$ is summarized as follows:

$$\begin{aligned} M \;:=\; & \texttt{hyp}\, v \mid \lambda(v:\tau).M \mid M \cdot M \mid \texttt{box}\;\; w\;\; M \mid \texttt{unbox-fetch}\;\; w\;\; M \\ & \mid \texttt{get-here}\;\; w\;\; M \mid \texttt{letdia-get}\;\; w\;\; [v_w := M]\;\; M \end{aligned}$$

$$\texttt{(hyp)}\;\frac{(v\colon A) \in \Gamma}{G \vdash (w,\Gamma) \vdash \texttt{hyp}\, v\colon A}$$

$$\texttt{(lambda)}\;\frac{\text{fresh}\;\; v_0 \qquad G \vdash (w,(v_0\colon A)\colon \Gamma) \vdash M\colon B}{G \vdash (w,\Gamma) \vdash \lambda(v_0 : A).M\colon A \to B}$$

$$\texttt{(appl)}\;\frac{G \vdash (w,\Gamma) \vdash M\colon A \to B \qquad G \vdash (w,\Gamma) \vdash N\colon A}{G \vdash (w,\Gamma) \vdash M \cdot N\colon B}$$

$$\texttt{(box)}\;\frac{\text{fresh}\;\; w_0 \qquad (w,\Gamma)\colon G \vdash (w_0,\emptyset) \vdash M\colon A}{G \vdash (w,\Gamma) \vdash \texttt{box}\;\; w_0\;\; M\colon \Box A}$$

$$\texttt{(unbox)}\;\frac{G \vdash (w,\Gamma) \vdash M\colon \Box A}{G \vdash (w,\Gamma) \vdash \texttt{unbox-fetch}\;\; w\;\; M\colon A}$$

$$\texttt{(unbox-fetch)}\;\frac{(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash M\colon \Box A}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash \texttt{unbox-fetch}\;\; w\;\; M\colon A}$$

$$\texttt{(here)}\;\frac{G \vdash (w,\Gamma) \vdash M\colon A}{G \vdash (w,\Gamma) \vdash \texttt{get-here}\;\; w\;\; M\colon \Diamond A}$$

$$\texttt{(get-here)}\;\frac{(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash M\colon A}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash \texttt{get-here}\;\; w\;\; M\colon \Diamond A}$$

$$\texttt{(letdia)}\;\frac{G \vdash (w,\Gamma) \vdash M\colon \Diamond A \qquad \begin{array}{c}\text{fresh}\;\; w_0, \text{fresh}\;\; v \\ (w_0,[v\colon A]), G \vdash (w,\Gamma) \vdash N\colon B\end{array}}{G \vdash (w,\Gamma) \vdash \texttt{letdia-get}\;\; w\;\; [v_{w_0} := M]\;\; N\colon B}$$

$$\texttt{(letdia-get)}\;\frac{\begin{array}{c}(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash M\colon \Diamond A \qquad \text{fresh}\;\; w_0, \text{fresh}\;\; v \\ (w_0,[v\colon A])\colon (w,\Gamma)\colon G \vdash (w',\Gamma') \vdash N\colon B\end{array}}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash \texttt{letdia-get}\;\; w\;\; [v_{w_0} := M]\;\; N\colon B}$$

Note that despite the fact that this type system is so similar to the one for $L_{IS5\text{-}LF}$, by adding host names in terms we are getting significantly more information. In $L_{IS5\text{-}LF}$ in order to decide whether or not contexts were swapped (e.g. in unbox rules) we needed to look at the whole proof tree. In $L_{IS5\text{-}Hyb}$ however, it is enough to know the name of current host: w - we can differentiate between, say, in-place `unbox-fetch` and host swapping one by simply comparing name of the current world w with name used in the term `unbox-fetch` w' M. If $w = w'$ then there was no exchange, otherwise there was one.

3.4.2 Operational semantics

Just like the typing system, the operational semantics of $L_{IS5\text{-}Hyb}$ mimics that of $L_{IS5\text{-}LF}$ – except making explicit use of host names.

The set of values is defined as:

val$(\lambda(A : v).M)$

val(`box` w_0 M)

if val(M), then val(`get-here` w M)

Note that reductions again take host name as a parameter.

$(\lambda(v : A).M) \cdot N \mapsto_w [N|v]M$

if $M \mapsto_w M'$, then $M \cdot N \mapsto_w M' \cdot N$

if $M \mapsto_w M'$, then `unbox-fetch` w $M \mapsto_{w'}$ `unbox-fetch` w M'

if $M \mapsto_w M'$, then `get-here` w $M \mapsto_{w'}$ `get-here` w M'

if $M \mapsto_w M'$,
then `letdia-get` w $[v_{w_0} := M]$ $N \mapsto_{w'}$
`letdia-get` w $[v_{w_0} := M']$ N

`unbox-fetch` w' (`box` w_0 M) $\mapsto_w \{w|w_0\}M$

if val(M),
then `letdia-get` w' $[v_{w_0} :=$ (`get-here` w'' $M)]$ $N \mapsto_w$
$[M|v_0]\{w''|w_0\}N$

As usual, proof trees for last two of these reduction rules look like this:

$$\Box E\,\dfrac{\Box I\,\dfrac{\overset{\mathcal{D}}{\text{fresh } w_0 \qquad (w',\Gamma')\colon (w,\Gamma)\colon G \vdash w_0\colon \emptyset \vdash M\colon A}}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash \texttt{box } w_0\ M\colon \Box A}}{(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash \texttt{unbox-fetch } w'\ (\texttt{box } w_0\ M)\colon A} \quad \Longrightarrow_R$$

$$\overset{\{w|w_0\}\mathcal{D}}{(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash \{w|w_0\}M\colon A}$$

where: $\{w|w_0\}((w',\Gamma')\colon (w,\Gamma)\colon G \vdash w_0\colon \emptyset \vdash M\colon A) = (w',\Gamma;)\colon G \vdash (w,\Gamma) \vdash \{w|w_0\}M\colon A$.

Let $G' = (w',\Gamma')\colon (w,\Gamma)\colon G$, $G'' = (w'',\Gamma'')\colon (w',\Gamma')\colon G$ and let fresh v and fresh w_0 in $\star$

$$\Diamond E\,\dfrac{\Diamond I\,\dfrac{\overset{\mathcal{D}}{G' \vdash (w'',\Gamma'') \vdash M\colon A}}{G' \vdash (w',\Gamma') \vdash \texttt{get-here } w''\ M\colon \Diamond A} \qquad \overset{\mathcal{E}}{(w_0,[v\colon A])\colon G'' \vdash (w,\Gamma) \vdash N\colon B}}{G'' \vdash (w,\Gamma) \vdash \texttt{letdia-get } w'\ [v_{w_0} := \texttt{get-here } w''\ M]\ N\colon B}\ \star$$

$$\Rightarrow_R \quad \overset{[\mathcal{D}|v]\{w'|w_0\}\mathcal{E}}{G'' \vdash (w,\Gamma) \vdash [M|v]\{w''|w_0\}N\colon B}$$

Note that just like in $\mathrm{L}_{\text{IS5-LF}}$ these trees are only one of several possibilities, as there are a two for `unbox-fetch` (`box` M) case and four in `letdia` (`get-here` M).

3.5 Progress and preservation

Our goal in this section is to justify that the three previously introduced languages behave the way we would expect them to. One measure we can take is checking if every term that we can type starting with an empty context, is either a value or there exists an evaluation step that we can make. This is usually referred to as progress.
The second property that we want to hold in a system is preservation of types. Say we have a term M of certain type A (again, with empty context). If this term M reduces then to M', then we want M' to be of type A as well.

What makes preservation and progress important? Progress ensures that we can continue evaluating every valid non-value expression – not necessarily to a value, but we will never be stuck with a computation due to a lack of

reduction rules. Preservation property allows us to be certain that evaluation will not change the type.

3.5.1 Progress

We have already given an informal definition of the progress property. The formal one of course depends on the language. We will use $\mathrm{L}_{\text{IS5-L}}$ as an example, but in all three languages, the proof follows the same pattern and is rather straightforward.

Theorem 3.1 (Progress)**.** If $\Omega; \emptyset \vdash M : A@w$ then either M is a value or $\exists M', M \mapsto_w M'$.

Proof. By induction on M:

- `hyp` v does not type in the empty environment, so it contradicts the precondition;
- $\lambda(v : A).M$ is a value;
- $M \cdot N$ is never a value, what it evaluates to depends on the result of induction hypothesis for M:
  - if M is a value, then (from the fact that it has a $\to$ type) it must be that $M = \lambda(v : A).M_0$, and β-reduction can occur, resulting in $M' = [v|N]M_0$
  - otherwise we have M_0 such that $M \mapsto_w M_0$ and we can continue evaluation of M under application, so $M' = M_0 \cdot N$;
- `box` w_0 M is a value;
- `unbox` M is similar to $M \cdot N$ in that the reduction result depends on the induction hypothesis for M - if it turns out to be a value – `box` w_0 M_0 expression, then we can reduce `unbox` M to $\{w|w_0\}M_0$. Otherwise we continue evaluation under `unbox` ;
- `here` M is the same; if M is a value then we reduce to `get` w (`here` M), otherwise continue evaluating under `here`;
- `letdia` $[v_{w_0} := M]$ N follows the same pattern as $M \cdot N$ and `unbox` M;
- `fetch` w M can be reduced either to M (when it is a value) or to `fetch` w M' (when $M \mapsto_w M'$);
- `get` w M may sometimes be a value - that is, when M is a value. Otherwise we continue execution under `get`.

3.5.2 Preservation

Again we will state it formally for $\mathrm{L}_{\text{IS5-L}}$:

(Preservation)**.** If $\Omega; \emptyset \vdash M : A@w$ and $M \mapsto_w M'$ then $\Omega; \emptyset \vdash M' : A@w$.

Preservation is a bit trickier to prove, we will therefore require a number of lemmas, in particular about type preservation through various substitutions – because of reductions such as $(\lambda(v : A).M) \cdot N \mapsto_w [N|v]M$ and `unbox` (`box` w_0 M) $\mapsto_w \{w|w_0\}M$. To prove those lemmas we will in turn require properties that allow us to extend the assumptions list or the set of known worlds while preserving the typing. We will start with these, as we will call them, weakening properties.

Property 3.5.2.1 (Context weakening)**.** If $\Omega; \Gamma \vdash M : A@w$ then for any Δ s.t. $\Delta \cap \Gamma = \emptyset$, we have $\Omega; \Gamma \cup \Delta \vdash M : A@w$.

Proof. Simple induction on the type derivation for M. We should note here that order of assumptions in context does not matter.

Property 3.5.2.2 (Known worlds weakening)**.** If $\Omega; \Gamma \vdash M : A@w$ then for any w_0 s.t. $w_0 \notin \Omega$, $w_0 : \Omega; \Gamma \vdash M : A@w$.

Proof. Simple induction on the type derivation for M.

Next, two results regarding types preserved by substitution.

Lemma 3.5.1. If $\Omega; (v : A@w'), \Gamma \vdash N : B@w$ and $\Omega; \emptyset \vdash M : A@w'$ then $\Omega; \Gamma \vdash [M|v]N : B@w$.

Proof. Induction on type derivation for N. One interesting case is `hyp` v' when $v = v'$. We then need to show that $\Omega; \Gamma \vdash M : B@w$. Uniqueness of variable names gives us $A = B$, $w = w'$. Then, using weakening variant from 3.5.2.1, we can remove context Γ and prove this case.
Note that for `box` and `letdia` cases, simple as they are, we actually need 3.5.2.2 to be able to use induction hypothesis.

Lemma 3.5.2. If $w_0 \in \Omega$ and $w_1 : \Omega; \Gamma \vdash M : A@w$ then $\Omega; \{w_0|w_1\}\Gamma \vdash \{w_0|w_1\}M : A@(\{w_0|w_1\}w)$.

Proof. Induction on type derivation for M. This proof involves a lot of careful case analysis, in particular for `fetch` and `get` cases, as the host name might have changed because of this substitution.

Finally, the sketch of Preservation proof:

Theorem 3.2 (Preservation)**.** If $\Omega; \emptyset \vdash M : A@w$ and $M \mapsto_w M'$ then $\Omega; \emptyset \vdash M' : A@w$.

Proof. Induction first on type derivation for M, then on reduction from M to M'.

- $(\lambda(v : A).M) \cdot N \mapsto_w [N|v]M$ from 3.5.1
- $\texttt{unbox}\ (\texttt{box}\ w_0\ M) \mapsto_w \{w|w_0\}M$ from 3.5.2
- $\texttt{letdia}\ [v_{w_0} := \texttt{get}\ w'\ (\texttt{here}\ M)]\ N \mapsto_w [M|v]\{w'|w_0\}N$ requires both 3.5.1 and 3.5.2, we also need to take into account that w may be equal to w' or not
- $\texttt{fetch}\ w'\ M \mapsto_w M$ knowing that $\mathrm{val}(M)$ we deduce $M = \texttt{box}\ \ w_0 M_0$; we can then choose w instead of w' in $\Box$-intro rule

The rest of the cases are simple and follow directly from induction hypothesis.

We have only talked about progress and preservation in $\mathrm{L}_{\text{IS5-L}}$. As it turns out, there aren't any differences when one wants to do the same in $\mathrm{L}_{\text{IS5-LF}}$ or $\mathrm{L}_{\text{IS5-Hyb}}$. In fact, these languages are similar enough to create translations between them.

Chapter 4

Relations between languages

> In this chapter we establish relationship between $L_{IS5\text{-}L}$, $L_{IS5\text{-}LF}$ and $L_{IS5\text{-}Hyb}$. We do so by providing type-preserving transformations between these languages.

Just by looking at type systems for three languages presented in the previous chapter, we can notice a lot of similarities. Our goal in this chapter is to formally describe these, either in terms of functional transformations or using relations between terms in different languages. We focus on both type and reduction preservation – as we will see it is not always possible to achieve both, though usually for technical reasons.
As much as we would like to provide a functional translation on terms for each pair of languages that interests us, it has proven to be infeasible. Note for example that moving from $L_{IS5\text{-}LF}$ to $L_{IS5\text{-}Hyb}$ requires adding context names to both terms and context, in a controlled manner. It is much more natural to express this translation by using a proof tree to reconstruct term from $L_{IS5\text{-}Hyb}$. Some of the difficulties come from our choice of implementation – these we will describe in full detail in Appendix A.

First however, let us explain the motivation for creating such transformations in the first place. As some of the proofs of language properties become technical, it is worth it to be able to prove a given property in one language and – through language transformations – extend its truthfulness to others. For example, proofs of normalization and termination properties seem to be less technically challenging in $L_{IS5\text{-}LF}$ than in any other language.
In addition, as all the languages use the same logic as their type system, we want to know exactly how close they are to one another. Perhaps at first $L_{IS5\text{-}L}$ and $L_{IS5\text{-}LF}$ do not seem identical, but through transformations between each of them and the in-between hybrid language, we can be more

precise in where these differences lie.

We have created hybrid language $\mathrm{L}_{\text{IS5-Hyb}}$ with a single goal in mind: we wanted for it to be immersed in both $\mathrm{L}_{\text{IS5-L}}$ and $\mathrm{L}_{\text{IS5-LF}}$. In this we have been successful, as both translations are done using easy to understand functions. We will begin by describing them, then move to other, more complex translations.

4.1 From $\mathrm{L}_{\text{IS5-Hyb}}$ to $\mathrm{L}_{\text{IS5-L}}$

As the relation between $\mathrm{L}_{\text{IS5-Hyb}}$ and $\mathrm{L}_{\text{IS5-LF}}$ is trivial, we will begin with a more interesting case. We aim at creating a pair of translation functions HybtoL – one for contexts translation, one for terms. We want the following conditions to be satisfied:

1 (Typing Preservation)**.** Let $\mathrm{HybtoL}\,((w, \Gamma) \colon G) = (\Omega, \Delta, w)$ and let M' be the result of $\mathrm{HybtoL}\,(M)$. Then from $G \vdash (w, \Gamma) \vdash_{\text{Hyb}} M \colon A$, it follows that $\Omega; \Delta \vdash_{\mathrm{L}} M' \colon A@w$

2 (Value Preservation)**.** For any term $M \in \mathrm{term}_{\text{Hyb}}$, if $\mathrm{val}_{\text{Hyb}}(M)$ then $\mathrm{val}_{\mathrm{L}}(\mathrm{HybtoL}\,(M))$.

For the next condition we require multi-step reductions ($\overset{+}{\mapsto}$). We define them inductively as:

If $M \mapsto M'$ then $M \overset{+}{\mapsto} M'$
If $M \mapsto M'$ and $M' \overset{+}{\mapsto} M''$ then also $M \overset{+}{\mapsto} M''$

3 (Reduction Preservation)**.** Assume that $M \mapsto_w M'$ in $\mathrm{L}_{\text{IS5-Hyb}}$ and let $\mathrm{HybtoL}\,(M) = N$, $\mathrm{HybtoL}\,(M') = N'$. Then $N \overset{+}{\mapsto}_w N'$ in $\mathrm{L}_{\text{IS5-L}}$.

The first condition states that typing is preserved through HybtoL translation, the second requires us to transform values to values, and the last talks about preservation of multi-step reduction. Of course, we would rather use a single-step reduction, but it is impossible, as for example in $\mathrm{L}_{\text{IS5-L}}$ we have separate unbox and fetch operators, whereas in $\mathrm{L}_{\text{IS5-Hyb}}$ we only have a combination of them, called `unbox-fetch` . It may therefore be the case that one-step reduction from $\mathrm{L}_{\text{IS5-Hyb}}$ is simulated by two reduction steps in $\mathrm{L}_{\text{IS5-L}}$ – in our example, one for `unbox` and one for `fetch`.

Function HybtoL for contexts rewriting is easy to create. We want to gather all context names (worlds) from the environment $(w, \Gamma) \colon G$ into Ω

and annotate every assumption from this environment with the appropriate world. We define such annotation for any $(w_0, \Gamma_0) \in (w, \Gamma) \colon G$ as follows:

$$\begin{aligned}&\text{annotateWorlds}(w_0, \emptyset) = \emptyset\\&\text{annotateWorlds}(w_0, (v \colon A) \colon \Gamma_0') = (v \colon A@w_0) \colon \text{annotateWorlds}(w_0, \Gamma_0')\end{aligned}$$

Next we simply concatenate all annotated contexts to create Δ[1].

$$\begin{aligned}&\Omega = \text{map fst } ((w, \Gamma) \colon G)\\&\Delta = \text{flatmap annotateWorlds } ((w, \Gamma) \colon G)\end{aligned}$$

Note that we did not change any names in the environment, just reordered the assumptions – that way our requirement of non-duplicating world names and variable names is maintained through this transformation.

Next let us create a translation for terms. It is fairly straightforward. The syntax we use here is Coq-like[2], with pattern matching on constructors.

$$\begin{aligned}&\text{HybtoL}\,(M_0) := \text{match } M_0 \text{ with}\\&\quad \mid \texttt{hyp}_{\text{Hyb}}\, v \Rightarrow \texttt{hyp}_{\text{L}}\, v\\&\quad \mid \lambda_{\text{Hyb}}(v : A).M \Rightarrow \lambda_{\text{L}}(v : A).(\text{HybtoL}\, M)\\&\quad \mid M \cdot_{\text{Hyb}} N \Rightarrow (\text{HybtoL}\, M) \cdot_{\text{L}} (\text{HybtoL}\, N)\\&\quad \mid \texttt{box}_{\text{Hyb}}\ w\ M \Rightarrow \texttt{box}_{\text{L}}\ w\ (\text{HybtoL}\, M)\\&\quad \mid \texttt{unbox-fetch}_{\text{Hyb}}\ w\ M \Rightarrow \texttt{unbox}_{\text{L}}\ (\texttt{fetch}_{\text{L}}\ w\ (\text{HybtoL}\, M))\\&\quad \mid \texttt{get-here}_{\text{Hyb}}\ w\ M \Rightarrow \texttt{get}_{\text{L}}\ w\ (\texttt{here}_{\text{L}}\ (\text{HybtoL}\, M))\\&\quad \mid \texttt{letdia-get}_{\text{Hyb}}\ w\ [v_{w_0} := M]\ N \Rightarrow\\&\qquad\qquad \texttt{letdia}_{\text{L}}\ [v_{w_0} := \texttt{get}_{\text{L}}\ w\ (\text{HybtoL}\, M)]\ (\text{HybtoL}\, N)\end{aligned}$$

We can immediately notice that HybtoL (`unbox-fetch` $w\ M$) along with HybtoL (`get-here` $w\ M$) and HybtoL (`letdia-get` $w\ [v_{w_0} := M]\ N$) could all be optimized into using `get` (or `fetch`) only when the world is really changing. Indeed, we could have kept current world as a parameter and rewrite e.g. HybtoL (`unbox-fetch` $w\ M, w$) to `unbox` (HybtoL (M, w)). This additional world parameter would, however, complicate reasoning about such function – in particular for recursive call within the `box` case, where current world should be fresh.
Our solution requires a possibly redundant use of get (or fetch) operator,

[1] flatmap is concat over map for any function that returns a list type
[2] To be exact this is the syntax for fixpoint in Coq

but avoids such problems. Additionally, note that the following proof tree (and similar ones for `get-here` and `letdia-get`) justifies such step, at least typing-wise:

$$\frac{\begin{array}{c}\mathcal{D}\\ \Omega;\Gamma \vdash_{\mathrm{L}} M : \Box A@w \end{array} \qquad w \in \Omega \text{ (as } M \text{ types in } w)}{\Omega;\Gamma \vdash_{\mathrm{L}} \texttt{fetch}\ w\ M : \Box A@w}$$

In order to show reduction preservation property we have mentioned at the beginning of this section, we will require these two lemmas regarding substitution and world renaming with relation to HybtoL :

Lemma 4.1.1. $[\mathrm{HybtoL}\,(M)|v](\mathrm{HybtoL}\,(N)) = \mathrm{HybtoL}\,([M|v]N))$.

Proof. Simple induction over N.

Lemma 4.1.2. $\{w|w'\}(\mathrm{HybtoL}\,(N)) = \mathrm{HybtoL}\,(\{w|w'\}N))$.

Proof. Simple induction over N.

We are now ready to show the preservation properties. We begin with typing preservation:

Theorem 4.1 (Typing Preservation)**.** Let $\mathrm{HybtoL}\,((w,\Gamma) : G) = (\Omega, \Delta, w)$ and let M' be the result of $\mathrm{HybtoL}\,(M)$. Then from $G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} M : A$, it follows that $\Omega;\Delta \vdash_{\mathrm{L}} M' : A@w$.

Proof. Induction on type derivation for M. We will cover only selected cases:

- $G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} \texttt{hyp}\, v : A$
  $M' = \texttt{hyp}\, v$ and from correctness of contexts rewriting, since $v : A \in \Gamma$ then also $v : A@w \in \Delta$. From the definition of set of known worlds Ω: $\Omega = \mathrm{map}\ \mathrm{fst}\ ((w,\Gamma) : G) = w : (\mathrm{map}\ \mathrm{fst}\ G)$, we conclude $w \in \Omega$.

- $G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} \lambda(v : A).M_0 : A \to B$
  In this case $M' = \lambda(v : A).(\mathrm{HybtoL}\,(M_0))$. For M' to have the type $A \to B$ we require that $\Omega;(v : A@w) : \Delta \vdash_{\mathrm{L}} \mathrm{HybtoL}\,(M_0) : B$. First, from the typing derivation of $\lambda(v : A).M_0$ we obtain that in the case of M_0, $G \vdash (w,(v,A) : \Gamma) \vdash_{\mathrm{Hyb}} M_0 : B$. To use the induction hypothesis it remains to check: $\mathrm{HybtoL}\,((w,(v : A) : \Gamma) : G) = (\Omega,(v : A@w) : \Delta, w)$.

- $G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} \texttt{box}\ \ w_0 M_0 : \Box A$
  $M' = \texttt{box}\ w_0\ (\mathrm{HybtoL}\,(M_0))$; for it to type we require that both $w \in \Omega$ and $w_0 : \Omega;\Delta \vdash_{\mathrm{L}} \mathrm{HybtoL}\,(M_0) : A$ hold. The first condition was covered in $\texttt{hyp}\, v$ case, the second is just as simple – we have $(w,\Gamma) : G \vdash (w_0,\emptyset) \vdash_{\mathrm{Hyb}} M_0 : A$, so we only have to check that indeed $\mathrm{HybtoL}\,((w_0,\emptyset) : (w,\Gamma) : G) = (w_0 : \Omega, \Delta, w)$. Then by induction hypothesis $\mathrm{HybtoL}\,(M_0)$ is typing.

– $(w', \Gamma') \colon G \vdash (w, \Gamma) \vdash_{\text{Hyb}} \texttt{get-here}\ w'\ M_0 \colon \Diamond A$
In this case $M' = \texttt{get}\ w'\ (\texttt{here}\ (\text{HybtoL}\,(M_0)))$. We also know that $(w, \Gamma) \colon G \vdash (w', \Gamma') \vdash_{\text{Hyb}} M_0 \colon A$. Next, by induction hypothesis we have: $\Omega; \Delta \vdash_{\text{L}} \text{HybtoL}\,(M_0) \colon A@w'$ – this in turn allows to conclude that $\Omega; \Delta \vdash_{\text{L}} \texttt{here}\ (\text{HybtoL}\,(M_0)) \colon \Diamond A@w'$. Finally, we can show that $w \in \Omega$, and therefore $\Omega; \Delta \vdash_{\text{L}} M' \colon \Diamond A@w$.

Theorem 4.2 (Value Preservation)**.** For all $M \in \text{term}_{\text{Hyb}}$, if $\text{val}_{\text{Hyb}}(M)$ then $\text{val}_{\text{L}}(\text{HybtoL}\,(M))$.

Proof. Case analysis on values from $\text{L}_{\text{IS5-Hyb}}$. Compare the values from $\text{L}_{\text{IS5-Hyb}}$ with the ones from $\text{L}_{\text{IS5-L}}$:

$\text{val}_{\text{Hyb}}(\lambda(A : v).M)$	$\text{val}_{\text{L}}(\lambda(A : v).M)$
$\text{val}_{\text{Hyb}}(\texttt{box}\ w_0\ M)$	$\text{val}_{\text{L}}(\texttt{box}\ w_0\ M)$
if $\text{val}_{\text{Hyb}}(M)$	if $\text{val}_{\text{L}}(M)$
then $\text{val}_{\text{Hyb}}(\texttt{get-here}\ w\ M)$	then $\text{val}_{\text{L}}(\texttt{get}\ w\ (\texttt{here}\ M))$

It is straightforward to notice that HybtoL preserves values.

Theorem 4.3 (Reduction Preservation)**.** Assume that $M \mapsto_w M'$ in $\text{L}_{\text{IS5-Hyb}}$ and let $\text{HybtoL}\,(M) = N$, $\text{HybtoL}\,(M') = N'$. Then $N \overset{+}{\mapsto}_w N'$ in $\text{L}_{\text{IS5-L}}$.

Proof. Simple induction on reduction $M \mapsto_w M'$. Note that in most cases, $\overset{+}{\mapsto}$ relation is being interpreted as one-step reduction – with the exceptions of `unbox-fetch`, `get-here` and `letdia-get`, where two steps may be required.

This concludes the translation from $\text{L}_{\text{IS5-Hyb}}$ to $\text{L}_{\text{IS5-L}}$.

4.2 From $\text{L}_{\text{IS5-Hyb}}$ to $\text{L}_{\text{IS5-LF}}$

Translation from $\text{L}_{\text{IS5-Hyb}}$ to $\text{L}_{\text{IS5-LF}}$ is by far the simplest of all – we just remove world names from both the environment and terms. Again, we want to preserve types, values, and reductions. The complete transformation is given below:

$$
\begin{aligned}
&\text{HybtoLF}\,((w,\Gamma)\colon G) = ((\text{map snd } G),\Gamma)\\
&\text{HybtoLF}\,(M_0) := \text{match } M_0 \text{ with}\\
&\quad \mid \mathtt{hyp}_{\text{Hyb}}\, v \Rightarrow \mathtt{hyp}_{\text{LF}}\, v\\
&\quad \mid \lambda_{\text{Hyb}}(v:A).M \Rightarrow \lambda_{\text{LF}}(v:A).(\text{HybtoLF } M)\\
&\quad \mid M \cdot_{\text{Hyb}} N \Rightarrow (\text{HybtoLF } M) \cdot_{\text{LF}} (\text{HybtoLF } N)\\
&\quad \mid \mathtt{box}_{\text{Hyb}}\ w_0\ M \Rightarrow \mathtt{box}_{\text{LF}}\ (\text{HybtoLF } M)\\
&\quad \mid \mathtt{unbox\text{-}fetch}_{\text{Hyb}}\ w\ M \Rightarrow \mathtt{unbox}_{\text{LF}}\ (\text{HybtoLF } M)\\
&\quad \mid \mathtt{get\text{-}here}_{\text{Hyb}}\ w\ M \Rightarrow \mathtt{here}_{\text{LF}}\ (\text{HybtoLF } M)\\
&\quad \mid \mathtt{letdia\text{-}get}_{\text{Hyb}}\ w\ [v_{w_0} := M]\ N \Rightarrow\\
&\qquad \mathtt{letdia}_{\text{LF}}\ [v := \text{HybtoLF } M]\ (\text{HybtoLF } N)
\end{aligned}
$$

Again, the context transformation preserves variable uniqueness.

The term substitution lemma and a variant of world substitution lemma are both given below.

Lemma 4.2.1. $[\text{HybtoLF}\,(M)|v](\text{HybtoLF}\,(N)) = \text{HybtoLF}\,([M|v]N)$.

Proof. Simple induction over N.

Note that there are no world names in $\text{L}_{\text{IS5-LF}}$, therefore world substitution does not change the end result of HybtoLF :

Lemma 4.2.2. $\text{HybtoLF}\,(N) = \text{HybtoLF}\,(\{w|w'\}N)$.

Proof. Simple induction over N.

Similarly to previous case, we are interested in preservation of types, values and reductions – this time, single-step ones.

Theorem 4.4 (Typing Preservation)**.** Let $M' = \text{HybtoLF}\,(M)$; then from $G \vdash (w,\Gamma) \vdash_{\text{Hyb}} M\colon A$, we can conclude that $(\text{map snd } G) \vdash \Gamma \vdash_{\text{LF}} M'\colon A$.

Proof. Induction on type derivation for M.

Theorem 4.5 (Value Preservation)**.** For all $M \in \text{term}_{\text{Hyb}}$, if $\text{val}_{\text{Hyb}}(M)$ then $\text{val}_{\text{LF}}(\text{HybtoLF}\,(M))$.

Proof. Again, comparing values in $\mathrm{L}_{\text{IS5-Hyb}}$ and in $\mathrm{L}_{\text{IS5-LF}}$ allows us to justify this theorem:

$\mathrm{val}_{\mathrm{Hyb}}(\lambda(A : v).M)$	$\mathrm{val}_{\mathrm{LF}}(\lambda(A : v).M)$
$\mathrm{val}_{\mathrm{Hyb}}(\texttt{box}\ w_0\ M)$	$\mathrm{val}_{\mathrm{LF}}(\texttt{box}\ M\ )$
if $\mathrm{val}_{\mathrm{Hyb}}(M)$	if $\mathrm{val}_{\mathrm{LF}}(M)$
then $\mathrm{val}_{\mathrm{Hyb}}(\texttt{get-here}\ w\ M)$	then $\mathrm{val}_{\mathrm{LF}}(\texttt{get-here}\ M\ )$

Theorem 4.6 (Reduction Preservation)**.** Assume that $M \mapsto_w M'$ in $\mathrm{L}_{\text{IS5-Hyb}}$ and let $\mathrm{HybtoLF}\,(M) = N$, $\mathrm{HybtoLF}\,(M') = N'$. Then $N \mapsto N'$ in $\mathrm{L}_{\text{IS5-LF}}$.

Proof. Simple induction on $M \mapsto_{\mathrm{Hyb}} M'$ using substitution lemmas mentioned above.

4.3 From $\mathrm{L}_{\text{IS5-L}}$ to $\mathrm{L}_{\text{IS5-Hyb}}$

The next two translations are incomplete – we can show that typing is preserved, that values match in a certain sense, but not that reductions behave the way we would expect them to. We will briefly discuss the encountered problems, but first let us describe the actual transformations from $\mathrm{L}_{\text{IS5-L}}$ to $\mathrm{L}_{\text{IS5-Hyb}}$.

Rewriting contexts between these two languages can be seen simply as bucket-sorting. For each context from Ω, we extract its assumptions from Γ. The world marked as current and its corresponding context will be used as the current context in $\mathrm{L}_{\text{IS5-Hyb}}$, whereas the rest will be in the background. Note that we require the current context to be known in Ω for this procedure to return a correct result.

Translation for terms is mostly straightforward, with two interesting cases: fetch and get operators. We do not have those in $\mathrm{L}_{\text{IS5-Hyb}}$ – instead we allow `unbox-fetch`, `get-here` and `letdia-get` to possibly change worlds. This forces us to use a more complicated rewrite, where we aim at finding a term with adequate conclusions and premises in its typing rules.

Compare `(get)` rule with its simulation in $\mathrm{L}_{\text{IS5-Hyb}}$:

$$\frac{\begin{array}{c}\mathcal{D}\\ \Omega;\Gamma \vdash_{\mathrm{L}} M\colon \Diamond A@w \end{array} \qquad w' \in \Omega}{\Omega;\Gamma \vdash_{\mathrm{L}} \texttt{get}\ \ w\ M\colon \Diamond A@w'}$$

$$\frac{\begin{array}{c}\mathcal{D}'\\ (w',\Gamma')\colon G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} M\colon \Diamond A \end{array} \qquad \begin{array}{c}\mathcal{E}\\ (w'',[v\colon A])\colon (w,\Gamma)\colon G \vdash (w',\Gamma') \vdash N\colon \Diamond A \end{array}}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash_{\mathrm{Hyb}} \texttt{letdia-get}\ \ w\ [v_{w''} := M]\ N\colon \Diamond A}$$

where $N = \texttt{get-here}\ \ w''\ (\texttt{hyp}\, v)$ and $\mathcal{E}$ is the following:

$$\frac{\dfrac{}{v\colon A \in [v\colon A]}}{(w',\Gamma')\colon (w,\Gamma)\colon G \vdash (w'',[v\colon A]) \vdash_{\mathrm{Hyb}} \texttt{hyp}\, v\colon A}}{(w'',[v\colon A])\colon (w,\Gamma)\colon G \vdash (w',\Gamma') \vdash_{\mathrm{Hyb}} \texttt{get-here}\ \ w''\ (\texttt{hyp}\, v)\colon \Diamond A}$$

So for a fresh v and w'', $\texttt{get}_{\mathrm{L}}\ \ w\ M_{\mathrm{L}}$ will be transformed into $\texttt{letdia-get}_{\mathrm{Hyb}}\ \ w\ [v_{w''} := M_{\mathrm{Hyb}}]\ (\texttt{get-here}_{\mathrm{Hyb}}\ \ w''\ (\texttt{hyp}_{\mathrm{Hyb}}\, v))$.

Similarly, for `fetch`:

$$\frac{\begin{array}{c}\mathcal{D}\\ \Omega;\Gamma \vdash_{\mathrm{L}} M\colon \Box A@w \end{array} \qquad w' \in \Omega}{\Omega;\Gamma \vdash_{\mathrm{L}} \texttt{fetch}\ \ w\ M\colon \Box A@w'}$$

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}'\\ (w',\Gamma')\colon G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} M\colon \Box A \end{array}}{(w,\Gamma)\colon (w',\Gamma')\colon G \vdash (w_0,\emptyset) \vdash_{\mathrm{Hyb}} \texttt{unbox-fetch}\ \ w\ M\colon A} \qquad \text{fresh } w_0}{(w,\Gamma)\colon G \vdash (w',\Gamma') \vdash_{\mathrm{Hyb}} \texttt{box}\ \ w_0\ \texttt{unbox-fetch}\ \ w\ M\colon \Box A}$$

For technical reasons, explained in full detail in Appendix A, we actually chose an alternative rewrite for `fetch`:

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}'\\ (w',\Gamma')\colon G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} M\colon \Box A \end{array}}{(w',\Gamma')\colon G \vdash (w,\Gamma) \vdash_{\mathrm{Hyb}} \texttt{get-here}\ \ w\ M\colon \Diamond\Box A} \qquad \mathcal{E}}{(w,\Gamma)\colon G \vdash w',\Gamma' \vdash_{\mathrm{Hyb}} \texttt{letdia-get}\ \ w\ [v_{w''} := \texttt{get-here}\ \ w\ M]\ N\colon \Box A}$$

where $N = \texttt{box}\ \ w_0\ (\texttt{unbox-fetch}\ \ w''\ (\texttt{hyp}\, v))$ and $\mathcal{E}$ is the following:

$$\frac{\dfrac{\dfrac{\dfrac{}{v\colon A \in [v\colon A]}}{(w_0,\emptyset)\colon (w',\Gamma')\colon (w,\Gamma)\colon G \vdash (w'',[v\colon \Box A]) \vdash_{\mathrm{Hyb}} \texttt{hyp}\, v\colon \Box A}}{(w'',[v\colon \Box A])\colon (w',\Gamma')\colon (w,\Gamma)\colon G \vdash (w_0,\emptyset) \vdash_{\mathrm{Hyb}} \texttt{unbox-fetch}\ \ w''\ (\texttt{hyp}\, v)\colon A}}{(w'',[v\colon \Box A])\colon (w,\Gamma)\colon G \vdash (w',\Gamma') \vdash_{\mathrm{Hyb}} \texttt{box}\ \ w_0\ (\texttt{unbox-fetch}\ \ w''\ (\texttt{hyp}\, v))\colon \Box A}$$

Another important difference between $\mathrm{L}_{\text{IS5-L}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$ is noticeable when comparing `unbox` with its equivalent, the in-place `unbox-fetch`. The

latter explicitly puts the name of the current world in the term, while the former does not. It enforces the function LtoHyb to take the current world as a parameter.

$$
\begin{array}{l}
\mathrm{LtoHyb}(M_0, w) := \text{match } M_0 \text{ with} \\
\quad \mid \mathtt{hyp}_{\mathrm{L}}\, v \Rightarrow \mathtt{hyp}_{\mathrm{Hyb}}\, v \\
\quad \mid \lambda_{\mathrm{L}}(v : A).M \Rightarrow \lambda_{\mathrm{Hyb}}(v : A).(\mathrm{LtoHyb}(M, w)) \\
\quad \mid M \cdot_{\mathrm{L}} N \Rightarrow (\mathrm{LtoHyb}(M, w)) \cdot_{\mathrm{Hyb}} (\mathrm{LtoHyb}(N, w)) \\
\quad \mid \mathtt{box}_{\mathrm{L}}\ w_0\ M \Rightarrow \mathtt{box}_{\mathrm{Hyb}}\ w_0\ (\mathrm{LtoHyb}(M, w_0)) \\
\quad \mid \mathtt{unbox}_{\mathrm{L}}\ M \Rightarrow \mathtt{unbox}_{\mathrm{Hyb}}\ (\mathrm{LtoHyb}(M, w)) \\
\quad \mid \mathtt{here}_{\mathrm{L}}\ M \Rightarrow \mathtt{here}_{\mathrm{Hyb}}\ (\mathrm{LtoHyb}(M, w)) \\
\quad \mid \mathtt{letdia}_{\mathrm{L}}\ [v_{w_0} := M]\ N \Rightarrow \\
\qquad \mathtt{letdia\text{-}get}_{\mathrm{Hyb}}\ w\ [v_{w_0} := \mathrm{LtoHyb}(M, w)]\ (\mathrm{LtoHyb}(N, w)) \\
\quad \mid \mathtt{fetch}_{\mathrm{L}}\ w'\ M \Rightarrow \\
\qquad \text{let } w_0 := \text{freshWorld in} \\
\qquad \text{let } w'' := \text{freshWorld in} \\
\qquad \text{let } v := \text{freshVar in} \\
\qquad \text{let } N := \mathtt{box}_{\mathrm{Hyb}}\ w_0\ (\mathtt{unbox\text{-}fetch}\ \ w''(\mathtt{hyp}\, v)) \text{in} \\
\qquad \mathtt{letdia\text{-}get}_{\mathrm{Hyb}}\ w'\ [v_{w''} :=]\ \mathtt{get\text{-}here}_{\mathrm{Hyb}}\ w'\ (\mathrm{LtoHyb}(M, w'))N \\
\quad \mid \mathtt{get}_{\mathrm{L}}\ w'\ M \Rightarrow \\
\qquad \text{let } w'' := \text{freshWorld in} \\
\qquad \text{let } v := \text{freshVar in} \\
\qquad \mathtt{letdia\text{-}get}_{\mathrm{Hyb}}\ w'\ [v_{w''} := \mathrm{LtoHyb}(M, w')] \\
\qquad\qquad\qquad (\mathtt{get\text{-}here}_{\mathrm{Hyb}}\ w''\ (\mathtt{hyp}\, v))
\end{array}
$$

Another difficulty arising from the use of worlds in LtoHyb function for terms is not as easy to solve as the `unbox` vs `unbox-fetch` case: our function does not preserve term substitution the way we might want it to.

For reduction preservation (e.g. β-reduction case) we need to have the following property:
$\mathrm{LtoHyb}([M|v]N, w) = [\mathrm{LtoHyb}(M, w)|v](\mathrm{LtoHyb}(N, w))$.
The problem with this equality is clear when considering translation of term $\mathtt{box}_{\mathrm{L}}\ w_0\ N$:

$$
\begin{array}{l}
\mathrm{LtoHyb}([M|v](\mathtt{box}_{\mathrm{L}}\ w_0\ N), w) = \\
\mathrm{LtoHyb}(\mathtt{box}_{\mathrm{L}}\ w_0\ [M|v]N, w) = \\
\mathtt{box}_{\mathrm{Hyb}}\ w_0\ (\mathrm{LtoHyb}([M|v]N, w_0))
\end{array}
$$

$$[\text{LtoHyb}(M, w)|v](\text{LtoHyb}(\texttt{box}_{\text{L}}\ w_0\ N, w)) =$$
$$[\text{LtoHyb}(M, w)|v](\texttt{box}_{\text{Hyb}}\ w_0\ (\text{LtoHyb}(N, w_0))) =$$
$$\texttt{box}_{\text{Hyb}}\ w_0\ ([\text{LtoHyb}(M, w)|v](\text{LtoHyb}(N, w_0)))$$

Note the term under `box` is $\text{LtoHyb}([M|v]N, w_0)$ in the first equation and $[\text{LtoHyb}(M, w)|v](\text{LtoHyb}(N, w_0))$ in the second. The property we have mentioned above cannot guarantee these two to be equal – not without additional assumption that for any w, w', $\text{LtoHyb}(M, w) = \text{LtoHyb}(M, w')$. We will call terms with such property w-stable. This stability requirement enforces in particular that M is not of the form $\texttt{unbox}\ M'$, $\texttt{letdia}\ [v_w := M']\ N'$ or $\texttt{here}\ M'$. For that reason we cannot prove any more than the following lemma for general term substitution:

Lemma 4.3.1. If C is w-stable term from $\text{L}_{\text{IS5-L}}$, then for any M, v, w it is the case that

$$\text{LtoHyb}([C|v]N, w) = [\text{LtoHyb}(C, w)|v](\text{LtoHyb}(N, w))$$

Proof. By induction on N.

For worlds renaming there are no such problems, therefore we have the following lemma:

Lemma 4.3.2. For any worlds w_0, w_1, w and any term $N \in \text{term}_{\text{L}}$:

$$\text{LtoHyb}(\{w_0|w_1\}N, \{w_0|w_1\}w) = \{w_0|w_1\}\text{LtoHyb}(N, w)$$

Proof. By induction on N.

As with all the other translations, we would like to prove type, value and reduction preservation. The first of those is standard:

Theorem 4.7 (Type preservation)**.** Let $\text{LtoHyb}(\Omega, \Gamma, w) = (G, \Delta, w)$. Assuming $\Omega; \Gamma \vdash_{\text{L}} M : A@w$, let us denote $\text{LtoHyb}(M, w)$ as N.
Then $G \vdash (w, \Delta) \vdash_{\text{Hyb}} N : A$.

Proof. By induction on type derivation for M.

- $\texttt{hyp}\, v$ case requires that translation of contexts is proper, that is if $(v : A@w) \in \Gamma$ then $(v : A) \in \Delta$ when Δ is the context named w, extracted from Γ;
- $\texttt{fetch}\ w'\ M$ and $\texttt{get}\ w'\ M$ cases were covered in the proof trees at the beginning of this section;

– the rest of the cases follow directly from induction hypothesis.

Next, value preservation ensures that every value is translated to a terminating term – that is, either to a value or to something reducible to a value in a finite number of reduction steps.

Theorem 4.8 (Value preservation)**.** For all $M \in \mathrm{term_L}$, if $\mathrm{val_L}(M)$, $M' = \mathrm{LtoHyb}(M, w)$; then $\exists V, M' \overset{*}{\mapsto}_w V \wedge \mathrm{val_{Hyb}}(V)$.

Proof. Induction on $\mathrm{val_L}(M)$. Compare values in these two languages:

$\mathrm{val_L}(\lambda(A : v).M)$	$\mathrm{val_{Hyb}}(\lambda(A : v).M)$
$\mathrm{val_L}(\texttt{box}\ w_0\ M)$	$\mathrm{val_{Hyb}}(\texttt{box}\ w_0\ M)$
if $\mathrm{val_L}(M)$	if $\mathrm{val_{Hyb}}(M)$
then $\mathrm{val_L}(\texttt{get}\ w\ (\texttt{here}\ M))$	then $\mathrm{val_{Hyb}}(\texttt{get-here}\ w\ M)$

For $\lambda(A : v).M$ and $\texttt{box}\ w_0\ M$ it is straightforward that values are preserved ($V = M'$). For $\texttt{get}\ w\ '(\texttt{here}\ M)$ we need a couple of reductions to reach the value – that is due to the way we rewrite `get`.

Let v and w_f be fresh variable and world, respectively. Then:
$\mathrm{LtoHyb}(\texttt{get}\ w\ '(\texttt{here}\ M), w) =$
$\texttt{letdia-get}\ w'\ [v_{w_f} := \mathrm{LtoHyb}(\texttt{here}\ M, w')]\ (\texttt{get-here}\ w_f\ (\texttt{hyp}\, v)) =$
$\texttt{letdia-get}\ w'\ [v_{w_f} := \texttt{get-here}\ w'\ (\mathrm{LtoHyb}(M, w'))]$
$(\texttt{get-here}\ w_f(\texttt{hyp}\, v))$.

We know that $\mathrm{val_L}(M)$ holds and from induction hypothesis we can find V such that $\mathrm{val_{Hyb}}(V)$ and $\mathrm{LtoHyb}(M, w') \overset{*}{\mapsto} V$.
Therefore $\texttt{get-here}\ w'\ (\mathrm{LtoHyb}(M, w')) \overset{*}{\mapsto}_{w'} \texttt{get-here}\ w'\ V$. This is also a value, by definition from $\mathrm{L_{IS5\text{-}Hyb}}$. Finally:
$\texttt{letdia-get}\ w'\ [v_{w_f} := \texttt{get-here}\ w'\ (\mathrm{LtoHyb}(M, w'))]$
$(\texttt{get-here}\ w_f\ (\texttt{hyp}\, v)) \overset{*}{\mapsto}_w$
$\texttt{letdia-get}\ w'\ [v_{w_f} := \texttt{get-here}\ w'\ V]\ (\texttt{get-here}\ w_f\ (\texttt{hyp}\, v)) \mapsto_w$
$[\texttt{get-here}\ w'\ V|v](\{w'|w_f\}(\texttt{get-here}\ w_f\ (\texttt{hyp}\, v))) =$
$[\texttt{get-here}\ w'\ V|v](\texttt{get-here}\ w'\ (\texttt{hyp}\, v)) =$
$\texttt{get-here}\ w'\ (\texttt{get-here}\ w'\ V)$.
This indeed is a value.

Finally, a remark on reduction preservation. As we have mentioned, the term substitution lemma is not as strong as we wanted, which in turn causes problems for example in the β-reduction preservation. We cannot show any form of reduction preservation using $\mathrm{LtoHyb}(M, w)$.
However, we can define a relation that mimics behavior of this function, except it is defined only between pairs of terms: $\mathrm{LtoHyb_R}(M_\mathrm{L}, M_\mathrm{Hyb})$. For terms like `unbox` or `here`, this relation accepts any world ($\mathrm{LtoHyb_R}(M, M') \to \forall w, \mathrm{LtoHyb_R}(\texttt{unbox}\ M, \texttt{unbox-fetch}\ w\ M')$. For $\mathrm{LtoHyb_R}(M, N)$ we can try to prove some variants of reduction preservation property:

Theorem 4.9 (Reduction preservation)**.** For any pair of terms M, M' that do not contain `fetch` or `get` and such that $M \mapsto_w M'$, we can find N and N' such that $\mathrm{LtoHyb_R}(M, N)$ and $\mathrm{LtoHyb_R}(M', N')$ and that $N \mapsto_w N'$.

Proof. Simple induction on reduction step $M \mapsto_w M'$.

An attempt to extend such theorem with terms containing `get` will require a weaker conclusion: If $M \mapsto M'$ then there exists N, N', N'' such that $\mathrm{LtoHyb_R}(M, N)$, $\mathrm{LtoHyb_R}(M', N')$ and $N \overset{*}{\mapsto} N''$, $N' \overset{*}{\mapsto} N''$. This change is a result of the variant of value preservation that we have.

Another extension is allowing `fetch` in the term from $\mathrm{L_{IS5\text{-}L}}$. The reason for `fetch` to be problematic is the case $\texttt{fetch}\ w'\ M$ with $\mathrm{val}(M)$. In $\mathrm{L_{IS5\text{-}L}}$ we reduce such term to M. After translation to $\mathrm{L_{IS5\text{-}Hyb}}$, we obtain a term that in many steps reduces also to a value – but a different one: $\texttt{box}\ w_0\ (\texttt{unbox-fetch}\ w'\ M')$ where $\mathrm{LtoHyb_R}(M, M)'$. As this latter term is in fact η-expansion of M', we can try to adapt the theorem to accept also such reductions as connected.

4.4 From $\mathrm{L_{IS5\text{-}LF}}$ to $\mathrm{L_{IS5\text{-}Hyb}}$

Finally, we want to talk about transition from $\mathrm{L_{IS5\text{-}LF}}$ to $\mathrm{L_{IS5\text{-}Hyb}}$. It is by far the most complicated of the four presented here – this is due to the fact that in order to create a correct term in $\mathrm{L_{IS5\text{-}Hyb}}$ in some cases we have to look at fragments of proof trees from $\mathrm{L_{IS5\text{-}LF}}$ – e.g. to know if $\texttt{unbox}_\mathrm{LF}\ M$ was local or did it change the current context.

We will begin as usual, by context transformation.

Suppose we have a multi-context environment G from $\mathrm{L_{IS5\text{-}LF}}$ that is correct (i.e. no variable is repeated). Let us take another multi-context environment G', this time from $\mathrm{L_{IS5\text{-}Hyb}}$. We will say that G' is compatible with

G when there are no repeated worlds in G' and $G \cong \text{map snd } G'$ – that is G is a permutation of contexts[3] from G', but without the names.
It is easy to see that any function assigning name to each context $\Gamma \in G$ without repeating these names, creates a compatible context.

Next, how do we rewrite the terms? There are three problematic cases: `unbox`, `here` and `letdia`. Their counterparts from $\text{L}_{\text{IS5-Hyb}}$ explicitly name the world that has been exchanged with the current one or use the name of current world to express no change. We therefore need a way of differentiating between these two rules in $\text{L}_{\text{IS5-LF}}$ – one exchanging worlds and the other leaving them as-is – when creating translation LFtoHyb:

$$\frac{G \vdash \Gamma \vdash_{\text{LF}} M : \Box A}{G \vdash \Gamma \vdash_{\text{LF}} \texttt{unbox}\ M : A} \qquad \frac{\Gamma' : G \vdash \Gamma \vdash_{\text{LF}} M : \Box A}{\Gamma : G \vdash \Gamma' \vdash_{\text{LF}} \texttt{unbox}\ M : A}$$

This means that the transformation will only work on terms that have a type. Note that it was not the case in any of the previous translations. Not only do we require $M \in \text{term}_{\text{LF}}$ to typecheck in order to produce its equivalent from $L_{\text{IS5-Hyb}}$, we also need to look at the details of the proof to make such translation.

Let us look at how we want LFtoHyb to work. In all cases assume:

$G_{\text{LF}} \cong \text{map snd } G_{\text{Hyb}}$,

w is a name of context Γ, w' is a name of context Γ' etc.,

w_0 is always a fresh world variable,

M_{Hyb} is the equivalent of M_{LF} in all the induction hypotheses, same for N_{LF} and N_{Hyb}.

Proof tree in $\text{L}_{\text{IS5-LF}}$	Result from $\text{L}_{\text{IS5-Hyb}}$
$\dfrac{(v : A) \in \Gamma}{G_{\text{LF}} \vdash \Gamma \vdash \texttt{hyp}_{\text{LF}}\, v : A}$	$\texttt{hyp}_{\text{Hyb}}\, v$
$\dfrac{\begin{matrix}\mathcal{D}\\ G_{\text{LF}} \vdash (v : A) : \Gamma \vdash M_{\text{LF}} : B\end{matrix}}{G_{\text{LF}} \vdash \Gamma \vdash \lambda_{\text{LF}}(v : A).M_{\text{LF}} : A \to B}$	$\lambda_{\text{Hyb}}(v : A).M_{\text{Hyb}}$
$\dfrac{\begin{matrix}\mathcal{D} \\ G_{\text{LF}} \vdash \Gamma \vdash M_{\text{LF}} : A \to B\end{matrix} \qquad \begin{matrix}\mathcal{E} \\ G_{\text{LF}} \vdash \Gamma \vdash N_{\text{LF}} : A\end{matrix}}{G_{\text{LF}} \vdash \Gamma \vdash M_{\text{LF}} \cdot_{\text{LF}} N_{\text{LF}} : A \to B}$	$M_{\text{Hyb}} \cdot_{\text{Hyb}} N_{\text{Hyb}}$

[3]Not only can contexts occur in a different order, but assumptions can be shuffled within the context

Proof tree in $\mathrm{L}_{\text{IS5-LF}}$	Result from $\mathrm{L}_{\text{IS5-Hyb}}$
$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma\colon G_{\mathrm{LF}} \vdash \emptyset \vdash M_{\mathrm{LF}}\colon A\end{array}}{G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{box}_{\mathrm{LF}}\ \ M_{\mathrm{LF}}\colon \Box A}$	$\mathtt{box}_{\mathrm{Hyb}}\ \ w_0\ M_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{c}\mathcal{D}\\ G_{\mathrm{LF}} \vdash \Gamma \vdash M_{\mathrm{LF}}\colon \Box A\end{array}}{G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{unbox}_{\mathrm{LF}}\ \ M_{\mathrm{LF}}\colon A}$	$\mathtt{unbox\text{-}fetch}_{\mathrm{Hyb}}\ \ w\ M_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma\colon G_{\mathrm{LF}} \vdash \Gamma' \vdash M_{\mathrm{LF}}\colon \Box A\end{array}}{\Gamma'\colon G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{unbox}_{\mathrm{LF}}\ \ M_{\mathrm{LF}}\colon A}$	$\mathtt{unbox\text{-}fetch}_{\mathrm{Hyb}}\ \ w'\ M_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{c}\mathcal{D}\\ G_{\mathrm{LF}} \vdash \Gamma \vdash M_{\mathrm{LF}}\colon A\end{array}}{G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{here}_{\mathrm{LF}}\ \ M_{\mathrm{LF}}\colon \Diamond A}$	$\mathtt{get\text{-}here}_{\mathrm{Hyb}}\ \ w\ M_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma\colon G_{\mathrm{LF}} \vdash \Gamma' \vdash M_{\mathrm{LF}}\colon A\end{array}}{\Gamma'\colon G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{here}_{\mathrm{LF}}\ \ M_{\mathrm{LF}}\colon \Diamond A}$	$\mathtt{get\text{-}here}_{\mathrm{Hyb}}\ \ w'\ M_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E}\\ G_{\mathrm{LF}} \vdash \Gamma \vdash M_{\mathrm{LF}}\colon \Diamond A & [(v\colon A)]\colon G_{\mathrm{LF}} \vdash \Gamma \vdash N_{\mathrm{LF}}\colon B\end{array}}{G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{letdia}_{\mathrm{LF}}\ \ [_v :=]\ M_{\mathrm{LF}} N_{\mathrm{LF}}\colon \Diamond A}$	$\mathtt{letdia\text{-}get}_{\mathrm{Hyb}}\ \ w\ [v_{w_0} := M_{\mathrm{Hyb}}]\ N_{\mathrm{Hyb}}$
$\dfrac{\begin{array}{cc}\mathcal{D} & \mathcal{E}\\ \Gamma\colon G_{\mathrm{LF}} \vdash \Gamma' \vdash M_{\mathrm{LF}}\colon \Diamond A & [(v\colon A)]\colon \Gamma'\colon G_{\mathrm{LF}} \vdash \Gamma \vdash N_{\mathrm{LF}}\colon B\end{array}}{\Gamma'\colon G_{\mathrm{LF}} \vdash \Gamma \vdash \mathtt{letdia}_{\mathrm{LF}}\ \ [v := M_{\mathrm{LF}}]\ N_{\mathrm{LF}}\colon \Diamond A}$	$\mathtt{letdia\text{-}get}_{\mathrm{Hyb}}\ \ w'\ [v_{w_0} := M_{\mathrm{Hyb}}]\ N_{\mathrm{Hyb}}$

Next we want to argue that this transformation preserves types and values.

Theorem 4.10 (Type preservation)**.** Let $\Gamma\colon G_{\mathrm{LF}}$ be compatible with $(w, \Gamma)\colon G_{\mathrm{Hyb}}$. Assume: $\dfrac{\begin{array}{c}\mathcal{D}\\ G_{\mathrm{LF}} \vdash \Gamma \vdash M_{\mathrm{LF}}\colon A\end{array}}{}$

Then $G_{\mathrm{Hyb}} \vdash (w, \Gamma) \vdash M_{\mathrm{Hyb}}\colon A$ where M_{Hyb} is obtained through transformation shown above.

Proof. Simple induction on type derivation for M_{LF}.

Theorem 4.11 (Value preservation)**.** For any M_{LF} that is a value, let $\Gamma : G_{\mathrm{LF}}$ be compatible with $(w, \Gamma) : G_{\mathrm{Hyb}}$ and assume that we have: $\mathcal{D}$ $G_{\mathrm{LF}} \vdash \Gamma \vdash M_{\mathrm{LF}} : A$

If M_{Hyb} is obtained through transformation mentioned previously, then $\mathrm{val}_{\mathrm{Hyb}}(M_{\mathrm{Hyb}})$.

Proof. Induction on $\mathrm{val}_{\mathrm{L}}(M)$. Compare values in these two languages:

$\mathrm{val}_{\mathrm{LF}}(\lambda(A : v).M)$	$\mathrm{val}_{\mathrm{Hyb}}(\lambda(A : v).M)$
$\mathrm{val}_{\mathrm{LF}}(\texttt{box}\ \ M)$	$\mathrm{val}_{\mathrm{Hyb}}(\texttt{box}\ w_0\ M)$
if $\mathrm{val}_{\mathrm{LF}}(M)$	if $\mathrm{val}_{\mathrm{Hyb}}(M)$
then $\mathrm{val}_{\mathrm{LF}}(\texttt{here}\ \ M)$	then $\mathrm{val}_{\mathrm{Hyb}}(\texttt{get-here}\ w\ M)$

The preservation of reductions theorem has not been formalized, but intuitively, the reductions for both languages are matching.

$\mathrm{L}_{\text{IS5-LF}}$	$\mathrm{L}_{\text{IS5-Hyb}}$
$\texttt{unbox}\ (\texttt{box}\ \ M) \mapsto M$	$\texttt{unbox-fetch}\ w'\ (\texttt{box}\ w_0\ M)$ $\mapsto_w \{w\|w_0\}M$
if $\mathrm{val}(M)$ then $\texttt{letdia}\ [v := \texttt{here}\ \ M]\ N \mapsto [M\|v]N$	if $\mathrm{val}(M)$ then $\texttt{letdia-get}\ w'\ [v_{w_0} := \texttt{get-here}\ w''\ M]$ N $\mapsto_w [M\|v_0]\{w''\|w_0\}N$
$(\lambda(v : A).M) \cdot N \mapsto [N\|v]M$	$(\lambda(v : A).M) \cdot N \mapsto_w [N\|v]M$
if $M \mapsto M'$ then $M \cdot N \mapsto M' \cdot N$	if $M \mapsto_w M'$ then $M \cdot N \mapsto_w M' \cdot N$
if $M \mapsto M'$ then $\texttt{unbox}\ M \mapsto \texttt{unbox}\ M'$	if $M \mapsto_w M'$ then $\texttt{unbox-fetch}\ w\ M$ $\mapsto_{w'} \texttt{unbox-fetch}\ \ wM'$
if $M \mapsto M'$ then $\texttt{here}\ M \mapsto \texttt{here}\ M'$	if $M \mapsto_w M'$ then $\texttt{get-here}\ w\ M \mapsto_{w'} \texttt{get-here}\ \ wM'$
if $M \mapsto M'$ then $\texttt{letdia}\ [v := M]\ N \mapsto$ $\texttt{letdia}\ [v := M']\ N$	if $M \mapsto_w M'$*then* $\texttt{letdia-get}\ w\ [v_{w_0} := M]\ N \mapsto_{w'}$ $\texttt{letdia-get}\ w\ [v_{w_0} := M']\ N$

This concludes language translations between $\mathrm{L}_{\text{IS5-L}}$, $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$.

Chapter 5

Termination of evaluation

> In this chapter we show termination of call-by-name evaluation for $\mathrm{L}_{\text{IS5-LF}}$. First we provide proof for sublanguage without the $\Diamond$ type. Next we discuss termination of full $\mathrm{L}_{\text{IS5-LF}}$ language. All proofs of termination in this chapter use Tait's method of logical relations as their base.

Operational semantics for languages described in Chapter 3 provide us with strategies of evaluation. In the previous chapter we have (partially) shown that reductions and values are compatible across three described languages, therefore there is a shared strategy of evaluation for all variants of L_{IS5}. The question we want to answer in this chapter is whether the chosen strategy is terminating. If that is the case, then by careful proof construction we can automatically extract the evaluator in OCaml from the proof in Coq.

Reason for choosing call by name evaluation strategy was the intended interpretation of the language, as described in Section 1 of Chapter 3. In this chapter we want to show that by evaluating a term that is typing in an empty environment we can eventually reach a term in the normal form (i.e. a value).

We define termination of term M as $M\downarrow := \exists V, M \overset{*}{\mapsto} V \wedge \mathrm{val}(V)$. Proof of this property will – in both presented variants – use logical relations (Tait's) method [8]. The first case described here, $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$, the proof will follow closely termination of simply-typed λ-calculus. The second one, termination of full $\mathrm{L}_{\text{IS5-LF}}$ language, uses slightly different logical relations, inspired by [11] and [3].

5.1 $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$

Termination of evaluation for $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$ (${}^{\backslash\Diamond}L_{\text{IS5-LF}}$) can be proven similarly to the same property for simply-typed λ-calculus. The only significant change is adding definition of reducibility for the $\Box$ type constructor.

Main goal of this section is to justify the following theorem:

(Termination of ${}^{\backslash\Diamond}L_{\text{IS5-LF}}$)**.** For every M, if $\text{emptyEquiv}\,(G) \vdash \emptyset \vdash M \colon A$[1], then $M\downarrow$.

When trying to prove such property directly, by induction on type derivation, we cannot get past the application case. We are unable to show $(M \cdot N)\downarrow$ having only assumptions: $M\downarrow$ and $N\downarrow$.

We therefore follow Tait's method and define a family of relations, one for each type, containing reducibility candidates. This definition is the following (where ι denotes base type):

$\text{Red}_\iota\, M := M\downarrow$

$\text{Red}_{A\to B}\, M := M\downarrow \wedge \forall N, \text{Red}_A\, N \to \text{Red}_B\, M \cdot N$

$\text{Red}_{\Box A}\, M := M\downarrow \wedge \text{Red}_A\, \texttt{unbox}\ M$

Note that we are allowed to require termination for reducible terms of complex types ($A \to B$ and $\Box A$) as we do not evaluate neither under λ expressions not under `box` – these are unconditional values in $\mathrm{L}_{\text{IS5-LF}}$[2] and therefore also in ${}^{\backslash\Diamond}L_{\text{IS5-LF}}$.

Let us first remark two observations for ${}^{\backslash\Diamond}L_{\text{IS5-LF}}$, that will make our reasoning simpler:

Observation (Redex uniqueness)**.** If $M \mapsto N$ and $M \mapsto N'$ in ${}^{\backslash\Diamond}L_{\text{IS5-LF}}$ then $N = N'$.

Observation. If $M \mapsto N$ then $M\downarrow$ if and only if $N\downarrow$.

Proof. It follows from the uniqueness of redex.

[1] As a reminder – $\text{emptyEquiv}\,(G)$ replaces each context from G with an empty context

[2] In contrast, in full $\mathrm{L}_{\text{IS5-LF}}$ the term `here` M is a value conditionally – namely only when M itself is a value.

We require all variants of the $\mathrm{Red}_A\, M$ relation to have the following properties:

Lemma 5.1.1 (CR1)**.** If $\mathrm{Red}_A\, M$ then $M\downarrow$.

Proof. By induction on A; it follows directly from the definition of $\mathrm{Red}_A\, M$ in each case.

Lemma 5.1.2 (CR2)**.** If $\mathrm{Red}_A\, M$ and $M \mapsto N$ then $\mathrm{Red}_A\, N$.

Proof. By induction on A.

Lemma 5.1.3 (CR3)**.** If $\mathrm{Red}_A\, N$ and $M \mapsto N$ then $\mathrm{Red}_A\, M$.

Proof. By induction on A.

Having 5.1.1, in order to show the termination property it suffices to argue that $\mathrm{emptyEquiv}\,(G) \vdash \emptyset \vdash M\colon A$ implies $\mathrm{Red}_A\, M$. This time we cannot work only on empty contexts, as in λ case we will need to use the induction hypothesis, and for typing of λ this has non-empty context. Therefore, we would like to prove the following:

(Reducibility Theorem)**.** For any given M, if $G \vdash \Gamma \vdash M\colon A$ then $\mathrm{Red}_A\, M$.

Unfortunately, the proof by induction on type derivation for M is again impossible. Consider the case $\lambda(v : A).M$. By the definition of $\mathrm{Red}_{A\to B}$ we need to show $\mathrm{Red}_B\, [N|v]M$. But from assumptions we only have $\mathrm{Red}_B\, M$ and $\mathrm{Red}_A\, N$ – and we know nothing about reducibility under substitution.

To remedy that we introduce a variant of term substitution: $[L|X]M$. This substitution is simultaneous, with L being a list of terms to be substituted and X being a list of variables. The final version of reducibility theorem is as follows:

Theorem 5.1 (Reducibility Theorem)**.** If $G \vdash \Gamma \vdash M\colon A$, X contains all free variables used in M and L contains reducible terms to be substituted for these variables, then $\mathrm{Red}_A\, [L|X]M$.

Proof. By induction on type derivation of M. In $\lambda(v : A).M$ and `box` $w_0\ M$ cases we use the property 5.1.3. The previously problematic part of λ case proof, showing reducibility of $[N|v]([L|X]M)$ in type B, is now solved simply by extending L and X by N and v, respectively.

Now it remains to prove the following:

Theorem 5.2 (Termination Theorem)**.** For any given M, if this term has a type in empty context: $\mathrm{emptyEquiv}\,(G) \vdash \emptyset \vdash M : A$, then $M \downarrow$.

Proof. It is enough to notice that as there are no assumptions in the environment, there can be no free variables. Therefore $X = \emptyset$, $L = \emptyset$ and from 5.1.1 and 5.1 we can conclude that indeed $M \downarrow$.

5.2 Complete $\mathrm{L}_{\text{IS5-LF}}$

In a sublanguage without $\Diamond$ everything was simple and straightforward. Do we expect the same from the full $\mathrm{L}_{\text{IS5-LF}}$? Judging from the rules for reducibility we have so far, any attempts to extend Red to capture $\Diamond A$ type as well would require using destructor of this type – `letdia`. Whether this is going to be problematic or not depends on the typing and reduction rules for the two operators that we add along with the $\Diamond$ type constructor. Let us remind them:

$$\texttt{(here)}\ \frac{G \vdash \Gamma \vdash M : A}{G \vdash \Gamma \vdash \texttt{here}\ M : \Diamond A} \qquad \texttt{(get-here)}\ \frac{\Gamma' : G \vdash \Gamma \vdash M : A}{\Gamma : G \vdash \Gamma' \vdash \texttt{here}\ M : \Diamond A}$$

$$\texttt{(letdia)}\ \frac{G \vdash \Gamma \vdash M : \Diamond A \quad \text{fresh } v_0 \quad [v_0 : A] : G \vdash \Gamma \vdash N : B}{G \vdash \Gamma \vdash \texttt{letdia}\ [v_0 := M]\ N : B}$$

$$\texttt{(letdia-get)}\ \frac{\Gamma : G \vdash \Gamma' \vdash M : \Diamond A \quad \text{fresh } v_0 \quad [v_0 : A] : \Gamma' : G \vdash \Gamma \vdash N : B}{\Gamma' : G \vdash \Gamma \vdash \texttt{letdia}\ [v_0 := M]\ N : B}$$

if $\mathrm{val}(M)$, then $\mathrm{val}(\texttt{here}\ M)$

if $\mathrm{val}(M)$, then $\texttt{letdia}\ [v := \texttt{here}\ M]\ N \mapsto [M|v]N$

if $M \mapsto M'$, then $\texttt{here}\ M \mapsto \texttt{here}\ M'$

if $M \mapsto M'$, then $\texttt{letdia}\ [v := M]\ N \mapsto \texttt{letdia}\ [v := M']\ N$

We do not expect `here` to give us much trouble – its typing is as simple as the one for `unbox`. `letdia` on the other hand uses arbitrary result type B. We do not know whether B is a smaller type than $\Diamond A$, so our usual inductive definition, along the lines of:
$\mathrm{Red}_{\Diamond A}\, M := \forall N\ \forall B\ \forall v,\ \text{fresh } v \to \mathrm{Red}_B\, N \to \mathrm{Red}_B\, \texttt{letdia}\ [v := M]\ N$
will not be proper!

The solution we propose uses the ideas described in [11] and [3]. First of these articles describes reducibility for strong normalization of a language

with operator `let`, similar to `letdia`, except with more concrete type of resulting expression N[3]:

$$(\texttt{let})\ \frac{M : \Diamond A \qquad N : \Diamond B}{\texttt{let}\ [v := M]\ N : \Diamond B}$$

The authors propose the following definition of reduciblility for $\Diamond A$ type:

$\mathrm{Red}_{\Diamond A}\, M$: Term M of type $\Diamond A$ is reducible if for all reducible continuations K, the application $K@M$ is strongly normalizing;

$\mathrm{Red}^{K}_{\Diamond A}\, K$: Continuation K accepting terms of type $\Diamond A$ is reducible if for all reducible V of type A, the application $K\ (\texttt{here}\ V)$ is strongly normalizing.

The continuations are defined there as $K :=\ = \mathrm{IdK}_\tau \mid K \circ (\texttt{let}\ [v := [\,]]\ N)$ and application of continuation is the following:

$\mathrm{IdK}_A@M = M$

$(K \circ (\texttt{let}\ [v := [\,]]\ N))@M = K@(\texttt{let}\ [v := M]\ N)$

We cannot use the proof method described there directly, as we have completely arbitrary result type of `letdia` while `let` always returns a computational ($\Diamond$) result . Nevertheless, we can change the above definitions into termination-expressing variant. They become:

$\mathrm{Red}_{\Diamond A}\, M := \forall\ K,\ \mathrm{Red}^{K}_{\Diamond A}\, K \to K@M \downarrow$

$\mathrm{Red}^{K}_{\Diamond A}\, K := \forall\ V,\ \mathrm{Red}_A\, V \to K@(\texttt{here}\ M) \downarrow$

Continuations and $K@M$ operation are defined similarly as before:

$K :=\ = \mathrm{IdK}_A \mid K \circ (\texttt{letdia}\ [v := [\,]]\ N)$

$\mathrm{IdK}_A@M = M$

$(K \circ (\texttt{letdia}\ [v := [\,]]\ N))@M = K@(\texttt{letdia}\ [v := M]\ N)$

With that, the proposed definitions of Red relation are complete. However, they turn out not to be enough to prove the following:

(Reducibility Theorem)**.** If $G \vdash \Gamma \vdash M : A$, X contains all free variables used in M and L contains reducible terms to be substituted for these variables, then $\mathrm{Red}_A\, [L|X]M$.

[3]In the original article the type of computation is denoted as TA, but for the sake of consistency we will continue to use $\Diamond A$.

The problem arises when showing the required property for `letdia`. This operator is not used directly in the definition of $\mathrm{Red}_{\Diamond A}\, M$, and the type of `letdia` is completely arbitrary.

What we can do in this case is use the approach described in [3] and, instead of showing reducibility, prove that $\forall\ K,\ \mathrm{Red}^K_A\, K \to K@([L|X]M)\downarrow$ (with the assumptions same as before). We will denote such property of a term as Q_A. The final variant of the reducibility theorem is the following:

(Reducibility Theorem)**.** If $G \vdash \Gamma \vdash M : A$, X contains all free variables used in M and L contains terms satysfying relation Q to be substituted for these variables, then $\mathrm{Q}_A\,([L|X]M)$.

We cannot leave the definitions of reducibility for types other than $\Diamond$ unchanged, rather we have to adapt them to the new goal similarly to $\mathrm{Red}_{\Diamond A}\, M$. First however, let us create continuations for the destructors of all the types:

$K ::= \mathrm{IdK}_A \mid K \circ (\texttt{letdia}\ \ [v := [\,]]\ N) \mid K \circ ([\,] \cdot N) \mid K \circ (\texttt{unbox}\ \ [\,])$

$\mathrm{IdK}_A@M = M$

$(K \circ (\texttt{letdia}\ \ [v := [\,]]\ N))@M = K@(\texttt{letdia}\ \ [v := M]\ N)$

$(K \circ ([\,] \cdot N))@M = K@(M \cdot N)$

$(K \circ (\texttt{unbox}\ \ [\,]))@M = K@(\texttt{unbox}\ \ M)$

This will allow us to prove e.g. $K@(\texttt{unbox}\ \ [L|X]M)\downarrow$ by extending the continuation K into $K \circ (\texttt{unbox}\ \ [\,])$ and using induction hypothesis on the remaining term, $[L|X]M$.

The reason for creating exactly these continuations is the order in which reductions are made. In particular for the continuations defined as above the following is true:

Property. For any M, N and K, if $M \mapsto N$ then $K@M \mapsto K@N$.

Next we have to define mutually recursive logical relations $\mathrm{Red}_A\, M$ and $\mathrm{Red}^K_A\, K$. Given that in the end, from reducibility theorem we want to prove "emptyEquiv $(G) \vdash \emptyset \vdash M : A \to M\downarrow$", we need to ensure that $\forall\ A,\ \mathrm{Red}^K_A\,(\mathrm{IdK}_A)$ holds, as this property will be required to make that final step.

We will begin with definitions for a basic type ι.

$\mathrm{Red}_\iota\, M := M\downarrow$

$\mathrm{Red}^K_\iota\, K := \forall\ V,\ V\downarrow \to K@V\downarrow$

The reducibility of terms is standard; for continuation the definition follows from the fact that, as mentioned above, we want identity continuation to be reducible.

Now let us move to definitions for $A \to B$ type. These relations are consistent with those from [3]:

$\mathrm{Red}_{A\to B}\, M := M\downarrow \wedge (\forall\ N,\ \mathrm{Q}_A\, N) \longrightarrow \mathrm{Q}_B\, (M \cdot N)$

$\mathrm{Red}^K_{A\to B}\, K := \forall\ M,\ \mathrm{Red}_{A\to B}\, M \to K@M\downarrow$

Note that despite $\mathrm{Red}^K_{A\to B}\, K$ calling $\mathrm{Red}_{A\to B}\, M$, this definition stays correct as in $\mathrm{Red}_{A\to B}\, M$ we use only recursive calls for smaller types.

For $\Box A$ the pattern is similar:

$\mathrm{Red}_{\Box A}\, M := M\downarrow \wedge \mathrm{Q}_A\, (\texttt{unbox}\ M)$

$\mathrm{Red}^K_{\Box A}\, K := \forall\ M,\ \mathrm{Red}_{\Box A}\, M \to K@M\downarrow$

Finally, it turns out that the previous definition of $\mathrm{Red}^K_{\Diamond A}\, K$ has stronger assumptions than we desire for $\texttt{here}\ \ M$ case (as now we aim at proving $\mathrm{Q}_A\, M$ rather than $\mathrm{Red}_A\, M$), we will therefore weaken them in the final version of the definition:

$\mathrm{Red}_{\Diamond A}\, M := \forall\ K,\ \mathrm{Red}^K_{\Diamond A}\, K \to K@M\downarrow = \mathrm{Q}_{\Diamond A}\, M$

$\mathrm{Red}^K_{\Diamond A}\, K := \forall\ V,\ \mathrm{Q}_A\, V \to K@(\texttt{here}\ \ M)\downarrow$

Now that the all reductions are all defined, let us first check the property of identity continuations we have mentioned previously:

Lemma 5.2.1. $\forall\ A,\ \mathrm{Red}^K_A\, (\mathrm{IdK}_A)$.

Proof. By induction on type A. The only non-trivial case is $\Diamond A$, the rest follow directly from the definitions of $\mathrm{Red}_A\, M$.
$\mathrm{Red}^K_{\Diamond A}\, \mathrm{IdK} = \forall\ V,\ \mathrm{Q}_A\, V \to \mathrm{IdK}@(\texttt{here}\ \ M)\downarrow$. We want to show that $\texttt{here}\ \ V\downarrow$ knowing that $V\downarrow$ (from $\mathrm{Q}_A\, V$). But in $\mathrm{L}_{\text{IS5-LF}}$ $(M\downarrow \to (\texttt{here}\ \ M)\downarrow)$, because we have defined value on `here` as "if $\mathrm{val}(M)$, then $\mathrm{val}(\texttt{here}\ \ M)$".

We are now ready to prove the reducibility theorem,

Theorem 5.3 (Reducibility Theorem)**.** If $G \vdash \Gamma \vdash M : A$, X contains all free variables used in M and L contains terms satisfying relation Q to be substituted for these variables, then $\mathrm{Q}_A\,([L|X]M)$.

Proof. Proof is by induction on type derivation of M. We will describe only selected cases.

- $M \cdot N$: $K@(([L|X]M) \cdot ([L|X]N)) \downarrow$
  Observe that by the definition of @ we have $K@(([L|X]M) \cdot ([L|X]N))$ $= (K \circ ([\,] \cdot ([L|X]N))@([L|X]M))$. By induction hypothesis on $[L|X]M$, such application terminates provided that $\mathrm{Red}^K_{A\to B}\,K \circ ([\,] \cdot ([L|X]N))$. But this follows directly from unfolding the definition.

- $\texttt{unbox}\ M$: $K@(\texttt{unbox}\ [L|X]M) \downarrow$
  Again, $K@(\texttt{unbox}\ [L|X]M) = (K \circ (\texttt{unbox}\ [\,]))@([L|X]M)$. Termination of the latter follows directly from the induction hypothesis.

- $\texttt{here}\ M$: $K@(\texttt{here}\ ([L|X]M)) \downarrow$
  From $\mathrm{Red}^K_{\Diamond A}\,K$ we get $\forall\ V,\ \mathrm{Q}_A\,V \to K@(\texttt{here}\ M) \downarrow$. It remains to show that $\mathrm{Q}_A\,([L|X]M)$. This is exactly the conclusion from the induction hypothesis.

Finally, the termination theorem is given below.

Theorem 5.4 (Termination Theorem)**.** For any given M, if this term has a type in empty context: $\mathrm{emptyEquiv}\,(G) \vdash \emptyset \vdash M : A$, then $M \downarrow$.

Proof. Same as in the sublanguage variant, $X = \emptyset$, $L = \emptyset$ – therefore $[L|X]M = M$. Next, by 5.2.1 we know that $\mathrm{Red}^K_A\,(\mathrm{IdK}_A)$, therefore from 5.3 we conclude that $M \downarrow$.

Chapter 6

Summary

In this thesis we have formalized languages for three variants of $\mathcal{ND}$ system for IS5. One of these systems, $\mathrm{L}_{\text{IS5-L}}$ along with the corresponding language was known beforehand, as the work of Tom Murphy VII et al. [14]. For the second one, the logic ($\mathrm{IS5}_{\mathrm{LF}}$) was described by Galmiche and Sahli [7], but the language itself is our contribution. Finally, $\mathrm{L}_{\text{IS5-Hyb}}$ is a new language for logic $\mathrm{IS5}_{\mathrm{Hyb}}$ (our variant of $\mathrm{IS5}_{\mathrm{LF}}$ which is not syntacticly pure) we have proposed in this thesis.

By creating an intermediate language between $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-L}}$, we were able to determine the connection between these two languages. We have shown how to transform terms from one language to the other in a type-preserving manner, using a two-step transformation with $\mathrm{L}_{\text{IS5-Hyb}}$ as an intermediate step. This immediately provides an equivalence of type systems ($\mathcal{ND}$ systems) for all three of these languages – that is, for three variants of IS5.

Further in Chapter 4 we have managed to immerse $\mathrm{L}_{\text{IS5-Hyb}}$ into both $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-L}}$ in a reduction-preserving manner. Moving in the other direction (from $\mathrm{L}_{\text{IS5-L}}$ and $\mathrm{L}_{\text{IS5-LF}}$ to $\mathrm{L}_{\text{IS5-Hyb}}$) while preserving reductions is an interesting future challenge, that would allow to identify syntactically pure $\mathrm{L}_{\text{IS5-LF}}$ with nicely interpretable $\mathrm{L}_{\text{IS5-L}}$.

We have discussed the termination of $\mathrm{L}_{\text{IS5-LF}}$ in Chapter 5. Using a simple adaptation of Tait's method, we provided termination result for $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$. We have also automatically extracted the evaluator for ${}^{\setminus\Diamond}L_{\text{IS5-LF}}$ from the proofs.

Finally, what we consider the biggest contribution of this thesis, we have shown termination of full $\mathrm{L}_{\text{IS5-LF}}$ using continuations.

We believe that an interesting extension of this thesis would be to consider termination of $L_{IS5\text{-}Hyb}$ and compare the obtained evaluator with both the one extractable from our proof of $L_{IS5\text{-}LF}$ termination and the one described in [13] for $L_{IS5\text{-}L}$.

It is worth noting that all the above mentioned contributions were formalized using Coq proof assistant. Most of this work has not been formalized ever before, and to the best of our knowledge none of it was defined in a proof assistant using the representation and system that we have chosen.

Appendix A

Implementation in COQ

Starting from chapter 3, all the formalizations and lemmas described in this thesis are implemented in the Coq proof assistant. This guarantees correctness of proofs and allows extracting evaluation algorithm from the formalization of termination proof. We will describe some of the technical aspects of our implementation, as well as a technique used for formalizing λ-calculus and its variants in a tool such as Coq, with no built-in α-conversion.

A.1 Representation of terms

When formalizing a programming language in a proof assistant such as Coq, we have to make a number of decisions, including the one regarding the way we represent terms and manipulate them – in particular, when dealing with variable binding. In "pen-and-paper" proofs, the standard approach is to use α-conversion when talking about variable binding – this allows us to consider $\lambda(v : A).v$ and $\lambda(v' : A).v'$ as α-equivalent. Coq is not equipped with tools that could help mimic this behavior (as is e.g. Twelf), so if we wanted to use α-conversion it would actually have to be implemented manually in some way. One possible approach is using variable names as usual and ensuring by a series of lemmas that α-equivalent terms are treated as such. What seems simple and natural to reason about is actually extremely complicated to implement.

Another possible implementation is de Bruijn indices. Instead of using named variables which have to match some binding (e.g. in λ), it uses indices indicating "how many bindings ago was this variable bound?". Compare these two formulas for a function that takes two arguments and returns the first one: $\lambda(x : A).\lambda(y : B).x$ is a standard term, $\lambda A.\lambda B.1$ is a version using de Bruijn indices.

This is without doubt something simple to implement in Coq. It is also fairly easy to use in case of term variables. One drawback of using this idea is that de Bruijn indices are far from natural to read and reason about. When formalizing a language we want to be as close to an actual language description, as possible – and with these, we are actually quite far away. Especially for languages like $\mathrm{L}_{\text{IS5-Hyb}}$ or $\mathrm{L}_{\text{IS5-L}}$, where not only do we have term variables, but also world variables, this becomes a serious problem when trying to relate traditional proof with its Coq code. Additional problems arise from using indices in formalizations of $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$. With de Bruijn indices, order of elements matters within each context. This makes merging two contexts together very unnatural and hard to implement.

Each of these representations for variables has its flaws. The combination of them as proposed in [1] tries to minimize them.

A.1.1 Locally nameless representation

In locally nameless (LN) representation terms under binding are still encoded using de Bruijn indices. However, variables from the context occurring free in the term are represented using names. For α-equivalence the problems arise only with binding, so we can safely use names for free variables. On the other hand, for bound variables de Bruijn indices remain the simplest to implement and since the indices are not used for free variable binding, order of elements in contexts does not matter anymore, and merging worlds is simple.

In order to use LN representation we also need a way of moving from bound variables (written with numbers and underline: $\underline{k}$) to free ones (usual variable names like $v, w, \ldots$ and overline: $\overline{v}$) and vice versa. Take for instance the original λ-introduction from $\mathrm{L}_{\text{IS5-LF}}$:

$$\texttt{(lambda)}\ \frac{\text{fresh } v_0 \qquad G \vdash (v_0 : A) : \Gamma \vdash M : B}{G \vdash \Gamma \vdash \lambda(v_0 : A).M : A \to B}$$

To represent it using LN we have to change this rule into one where term under λ does not use a named variable anymore, but rather an index. We define two operations dual to each other: opening and closing the term. Opening ($M^{\overline{x}}$) replaces last introduced bound variable with a free variable, closing (${}^{\backslash\overline{x}}M$) does the reverse. For example: $(\texttt{hyp}\ \underline{0}A)^{\overline{x}} = \texttt{hyp}\ \overline{x}A$, ${}^{\backslash\overline{x}}(\texttt{hyp}\ \overline{x}A) = \texttt{hyp}\ \underline{0}A$

We can think of opening and closing operations in terms of substitution. Opening can then be expressed as simply $M^{\overline{x}} = [\texttt{hyp}\ \overline{x}A|\underline{0}]M$, closing

similarly as ${}^{\backslash\overline{x}}M = [\texttt{hyp}\,\underline{0}A|\overline{x}]M$[1] Note that substitution accepts both free and bound variables as arguments. For bound variables, whenever moving through a variable binder like λ or `letdia`, we increase the index of substituted variable, like so:
$(\lambda A.\texttt{hyp}\,\underline{1}A)^{\overline{x}} = [\texttt{hyp}\,\overline{x}A|\underline{0}](\lambda A.\texttt{hyp}\,1A) = \lambda A.([\texttt{hyp}\,\overline{x}A|\underline{1}](\texttt{hyp}\,\underline{1}A)) =$
$\lambda A.\texttt{hyp}\,\overline{x}A$

New λ-introduction rule in $\mathrm{L}_{\text{IS5-LF}}$ will look like this:

$$\texttt{(lambda)}\ \frac{\text{fresh } v_0 \qquad G \vdash (v_0 \colon A) \colon \Gamma \vdash M^{\overline{v_0}} \colon B}{G \vdash \Gamma \vdash \lambda A.M \colon A \to B}$$

One might notice that our example with λ-opening used a term we would not consider proper: in the body of $\lambda A.\texttt{hyp}\,\underline{1}B$ we use variable bound two bindings away ($\underline{1}$), but the term has only one binder (λ). This leads to the definition of *locally closed* terms: terms that use bound variables only after their binder. In particular, $\lambda A.\texttt{hyp}\,\underline{1}B$ is not locally closed, but $\lambda A.\texttt{hyp}\,\underline{0}A$ is.

For two other languages, $\mathrm{L}_{\text{IS5-Hyb}}$ and $\mathrm{L}_{\text{IS5-L}}$, we use not only term variables, but also variables for worlds. We define similar opening and closing operations for them, again defined in terms of substitution as $t^{\overline{w}} = \{\overline{w}|\underline{0}\}t$ and ${}^{\backslash\overline{w}}t = \{\underline{0}|\overline{w}\}t$. We change substitution to accept both bound and free world variables and make a shift for bound variables whenever moving through a binder – in this case, `box` or `letdia`. Finally, we define the second version of locally closed terms - this time with respect to world variables.

In our formalization we hardly ever use term closing, but term opening (both with world and term variables) is used extensively. Additionally, many of the lemmas we aim at proving in Coq require that terms are locally closed. Below we provide sample implementations of term and world substitution, as well as both definitions of locally closed terms – all are given for $\mathrm{L}_{\text{IS5-Hyb}}$ case, but they are similar in other languages.

We begin with a shifting operation, defined for both free and bound variables. For bound variables (be it world or term ones), we increase the binding by one, for free variables we make no change.

[1] We will actually abuse this notation and allow opening with arbitrary term: $M^C = [C|\underline{0}]M$. For variables we will use $M^{\overline{x}}$ as a short form of $M^{\texttt{hyp}\,\overline{x}_A}$.

$$
\begin{aligned}
x^{\uparrow} := \text{match } x \text{ with} \\
&\mid \overline{v} \Rightarrow \overline{v} \\
&\mid \underline{k} \Rightarrow \underline{k+1}
\end{aligned}
$$

The term substitution is defined as the following:

$$
\begin{aligned}
&[C_0|x_0]M_0 := \text{match } M_0 \text{ with} \\
&\quad \mid \texttt{hyp}\, xA \Rightarrow \text{if } x_0 = x, \text{ then } C_0 \text{ else } \texttt{hyp}\, xA \\
&\quad \mid \lambda A.M \Rightarrow \lambda A.([C_0|x_0{}^{\uparrow}]M) \\
&\quad \mid M \cdot N \Rightarrow ([C_0|x_0]M) \cdot ([C_0|x_0]N) \\
&\quad \mid \texttt{box}\, M \Rightarrow \texttt{box}\, ([C_0|x_0]M) \\
&\quad \mid \texttt{unbox-fetch}\ w\ M \Rightarrow \texttt{unbox-fetch}\ w\ ([C_0|x_0]M) \\
&\quad \mid \texttt{get-here}\ w\ M \Rightarrow \texttt{get-here}\ w\ ([C_0|x_0]M) \\
&\quad \mid \texttt{letdia-get}\ {}^{A}w[M]N \Rightarrow \texttt{letdia-get}\ {}^{A}w[[C_0|x_0]M]([C_0|x_0{}^{\uparrow}]N)
\end{aligned}
$$

Note that world substitution shifts both sides of $\{w_0|w_1\}$:

$$
\begin{aligned}
&\{w_0|w_1\}M_0 := \text{match } M_0 \text{ with} \\
&\quad \mid \texttt{hyp}\, xA \Rightarrow \texttt{hyp}\, xA \\
&\quad \mid \lambda A.M \Rightarrow \lambda A.(\{w_0|w_1\}M) \\
&\quad \mid M \cdot N \Rightarrow (\{w_0|w_1\}M) \cdot (\{w_0|w_1\}N) \\
&\quad \mid \texttt{box}\, M \Rightarrow \texttt{box}\, (\{w_0{}^{\uparrow}|w_1{}^{\uparrow}\}M) \\
&\quad \mid \texttt{unbox-fetch}\ w\ M \Rightarrow \texttt{unbox-fetch}\ (\{w_0|w_1\}w)\ (\{w_0|w_1\}M) \\
&\quad \mid \texttt{get-here}\ w\ M \Rightarrow \texttt{get-here}\ (\{w_0|w_1\}w)\ (\{w_0|w_1\}M) \\
&\quad \mid \texttt{letdia-get}\ {}^{A}w[M]N \Rightarrow \\
&\qquad \texttt{letdia-get}\ {}^{A}(\{w_0|w_1\}w)[\{w_0|w_1\}M](\{w_0{}^{\uparrow}|w_1{}^{\uparrow}\}N)
\end{aligned}
$$

We finish this part by giving definitions of locally closed terms. $\mathrm{LC_t}\, M$ denotes that term M is closed with respect to term variables, $\mathrm{LC_w}\, M$ states the same for world variables.

$$
\begin{aligned}
&\mathrm{LC_t}(n)M_0 := \text{match } M_0 \text{ with}\\
&\quad \mid \texttt{hyp}\,\overline{v}A \Rightarrow \texttt{true}\\
&\quad \mid \texttt{hyp}\,\underline{k}A \Rightarrow \text{if } k < n, \text{ then } \texttt{true} \text{ else } \texttt{false}\\
&\quad \mid \lambda A.M \Rightarrow \mathrm{LC_t}(n+1)M\\
&\quad \mid M \cdot N \Rightarrow \mathrm{LC_t}(n)M \wedge \mathrm{LC_t}(n)N\\
&\quad \mid \texttt{box}\,M \Rightarrow \mathrm{LC_t}(n)M\\
&\quad \mid \texttt{unbox-fetch}\ \ w\ M \Rightarrow \mathrm{LC_t}(n)M\\
&\quad \mid \texttt{get-here}\ \ w\ M \Rightarrow \mathrm{LC_t}(n)M\\
&\quad \mid \texttt{letdia-get}\ {}^{A}w[M]N \Rightarrow \mathrm{LC_t}(n)M \wedge \mathrm{LC_t}(n+1)N\\
&\mathrm{LC_t}\,M = \mathrm{LC_t}(0)M
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{LC_w}(n)M_0 := \text{match } M_0 \text{ with}\\
&\quad \mid \texttt{hyp}\,vA \Rightarrow \texttt{true}\\
&\quad \mid \lambda A.M \Rightarrow \mathrm{LC_w}(n)M\\
&\quad \mid M \cdot N \Rightarrow \mathrm{LC_w}(n)M \wedge \mathrm{LC_w}(n)N\\
&\quad \mid \texttt{box}\,M \Rightarrow \mathrm{LC_w}(n+1)M\\
&\quad \mid \texttt{unbox-fetch}\ \ w\ M \Rightarrow \mathrm{LC_w}(n)M\\
&\quad \mid \texttt{get-here}\ \ w\ M \Rightarrow \mathrm{LC_w}(n)M\\
&\quad \mid \texttt{letdia-get}\ {}^{A}w[M]N \Rightarrow \mathrm{LC_w}(n)M \wedge \mathrm{LC_w}(n+1)N\\
&\mathrm{LC_w}\,M = \mathrm{LC_w}(0)M
\end{aligned}
$$

A.1.2 Cofinite quantification

Another concept mentioned in Charguéraud's et al. article [1] that we have used in formalization of variants of $\mathrm{L_{IS5}}$ is cofinite quantification. It is used to strengthen the induction principle on typing derivation.

After changes mentioned in the previous section, the rule for λ-introduction is the following:

$$
\texttt{(lambda)}\ \frac{\text{fresh } v_0 \qquad G \vdash (v_0 \colon A) \colon \Gamma \vdash M^{\overline{v_0}} \colon B}{G \vdash \Gamma \vdash \lambda A.M \colon A \to B}
$$

What is of interest to us now is the "fresh v_0" statement. What we usually mean by freshness is simply $v_0 \notin \mathrm{FV}(M)$. Sometimes such assumption is weak – not necessarily weak enough to make some property not true, but

enough to make it very challenging to formalize the proof, i.e. requiring renaming of variables.

To overcome such difficulties we use alternative approach – cofinite quantification. Instead of showing that some property holds for a given x outside of $\mathrm{FV}(M)$, we want to state that it holds for all $x \notin L$ for some finite set L. This changes our λ-introduction rule into:

$$\texttt{(lambda)}\ \frac{\forall v_0 \notin L, G \vdash (v_0 : A) : \Gamma \vdash M^{\overline{v_0}} : B}{G \vdash \Gamma \vdash \lambda A.M : A \to B}$$

The equivalence of cofinite definitions with their more standard counterparts for λ-calculus is shown in [1].

A.2 Languages for IS5

Languages used throughout this thesis – $\mathrm{L}_{\text{IS5-L}}$, $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$ – were all formalized using the representation of terms mentioned in the previous section. In this and the subsequent sections we would like to discuss the changes between description and actual formalization.

A.2.1 Environment representation

We have not yet mentioned how environments are encoded. For all the languages, a single context is represented as a list modulo permutation. For that we use permutations from `tlc` library, extended with some additional lemmas. To express that one context is a permutation of the other we use $\Gamma \approx \Gamma'$ notation.
Since in $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$ there are multiple contexts within an environment, we have another notion of permutation. $G \cong G'$ expresses the fact that two multi-context environments are a permutation of each other. The order of contexts in G and G' is irrelevant, as is order of assumption within one context. In $\mathrm{L}_{\text{IS5-LF}}$ it is therefore the case that $\Gamma \approx \Gamma' \to \Gamma : G \cong \Gamma' : G$; similar property in $\mathrm{L}_{\text{IS5-Hyb}}$ looks like this: $\Gamma \approx \Gamma' \to (w, \Gamma) : G \cong (w, \Gamma') : G$. We may think of $\cong$ as a "two-level" permutation.
In $\mathrm{L}_{\text{IS5-L}}$ the Ω set (containing all the known worlds) is also formalized as a list modulo permutation.

Another aspect of environment representation is its correctness – that is, our requirement that no variable (neither world not term one) can be repeated. We will use a predicate like $\mathrm{Ok}\,(\Omega, \Gamma)$, or $\mathrm{Ok}\,(G)$ to express that.

Finally, for properties like preservation, we need to be able to find an environment containing no assumptions at all, but with all the worlds intact. This we will denote as emptyEquiv (G). As for $\mathrm{L}_{\text{IS5-L}}$ this is simply setting $\Gamma = \emptyset$, there we will omit the use of such function.

A.2.2 $\mathbf{L}_{\text{IS5-L}}$

Differences between description and implementation of type system for $\mathrm{L}_{\text{IS5-L}}$ come from the change in term representation and follow the scheme presented for λ in previous section. All the rules below are also extended with a requirement $\mathrm{Ok}\,(\Omega, \Gamma)$ and all occurrences of variables in terms are now free variables ($\overline{v}$ instead of v, $\overline{w}$ instead of w).

$$\frac{\text{fresh } v_0 \qquad \Omega; (v_0 \colon A@w) \colon \Gamma \vdash M \colon B@w}{\Omega; \Gamma \vdash \lambda(v_0 : A).M \colon (A \to B)@w} \quad \to$$

$$\frac{\forall v_0 \notin L,\ \Omega; (v_0 \colon A@w) \colon \Gamma \vdash M^{\overline{v_0}} \colon B@w}{\Omega; \Gamma \vdash \lambda A.M \colon (A \to B)@w}$$

$$\frac{w \in \Omega \qquad \text{fresh } w_0 \qquad w_0 \colon \Omega; \Gamma \vdash M \colon A@w_0}{\Omega; \Gamma \vdash \texttt{box}\ w_0\ M \colon \Box A@w} \quad \to$$

$$\frac{w \in \Omega \qquad \forall w_0 \notin L,\ w_0 \colon \Omega; \Gamma \vdash M^{\overline{w_0}} \colon A@w_0}{\Omega; \Gamma \vdash \texttt{box}\, M \colon \Box A@w}$$

$$\frac{\Omega; \Gamma \vdash M \colon \Diamond A@w \qquad \begin{array}{c}\text{fresh } w_0, \text{fresh } v \\ w_0 \colon \Omega; (v \colon A@w_0) \colon \Gamma \vdash N \colon B@w\end{array}}{\Omega; \Gamma \vdash \texttt{letdia}\ [v_A := w_0]\ MN \colon B@w} \quad \to$$

$$\frac{\Omega; \Gamma \vdash M \colon \Diamond A@w \qquad \forall w_0 \notin L_w,\ \forall v \notin L_t,\ w_0 \colon \Omega; (v \colon A@w_0) \colon \Gamma \vdash (N^{\overline{w_0}})^{\overline{v}} \colon B@w}{\Omega; \Gamma \vdash \texttt{letdia}\, [A]MN \colon B@w}$$

Typing is implemented in Coq as an inductive predicate; we will only give a snippet with two cases:

```
Inductive types_L:
    worlds_L -> ctx_L -> te_L -> ty -> var -> Prop :=

| t_hyp_L: forall Omega Gamma v A w
```

```
  (Ok: ok_L Omega Gamma)
  (World: Mem w Omega)
  (HT: Mem (w, v, A) Gamma),
  Omega; Gamma |- hyp_L (fte v) ::: A @ w

| t_lam_L: forall L Omega Gamma w A B M
  (Ok: ok_L Omega Gamma)
  (HT: forall x, x \notin L ->
    Omega; (w, x, A)::Gamma |-
           (M ^t^ (hyp_L (fte x))) ::: B @ w),
  Omega; Gamma |- lam_L A M ::: A ---> B @ w

(...)

where " Omega ';' Gamma '|-' M ':::' A '@' w " :=
    (types_L Omega Gamma M A w): labeled_is5_scope.
```

The operational semantics do not differ much from the description either. We only add a requirement that terms are locally closed: $\mathrm{LC_t}\, M$, $\mathrm{LC_w}\, M$. Reduction step, $M \mapsto_w N$ is again an inductive predicate, same as $\mathrm{val}(M)$. These are again small snippets from the actual implementation:

```
Inductive value_L: te_L -> Prop :=
| val_lam_L: forall A M, value_L (lam_L A M)
| val_box_L: forall M, value_L (box_L M)
| val_get_here_L: forall M w (HT: value_L M),
    value_L (get_L w (here_L M)).

Inductive step_L: te_L * vwo -> te_L * vwo -> Prop :=
| red_appl_lam_L: forall A M N w,
   lc_w_L M -> lc_t_L (M ^t^ N) ->
   lc_w_L N -> lc_t_L N ->
   (appl_L (lam_L A M) N, w) |-> (M ^t^ N, w)
| red_unbox_box_L: forall M w,
   lc_t_L M -> lc_w_L (M ^w^ w) ->
   (unbox_L (box_L M), w) |-> (M ^w^ w, w)
| red_letd_get_here_L: forall M N w w',
   lc_t_L M -> lc_w_L M ->
   lc_t_L (N ^t^ M) -> lc_w_L (N ^w^ w') -> value_L M ->
   (letd_L (get_L w' (here_L M)) N, w) |-> ((N ^w^ w') ^t^ M, w)
| red_appl_L: forall M N M' w (HRed: (M, w) |-> (M', w)),
   lc_t_L M -> lc_w_L M ->
```

```
   lc_t_L N -> lc_w_L N ->
   (appl_L M N, w) |-> (appl_L M' N, w)
(...)
where " M |-> N " := (step_L M N ) : labeled_is5_scope.
```

Proof of the progress property remains straightforward – we have not changed anything in the definition of reduction, so there is no reason why it should be any different. The preservation proof has changed a little bit, as we use the locally nameless representation and cofinite quantification. In particular for variable-binding terms, λ, `box` and `letdia`, we now have a different-looking induction hypothesis.
The lemma that we are proving is the following:

```
Lemma PreservationL:
forall Omega M N A w w'
  (HType: Omega; nil |- M ::: A@w)
  (HStep: (M, fwo w) |-> (N,fwo w')),
  Omega; nil |- N ::: A@w'.
```

Let us discuss only the β-reduction case.
We have to show $\Omega; \emptyset \vdash M^N : B@w'$. From typing of $\lambda A.M$ we can conclude that $\forall v \notin L,\ \Omega; (v : A@w') : \Gamma \vdash M^{\overline{v}} : B@w'$.
It suffices to express M^N as a combination of two substitutions using a fresh variable as the "in between" argument. $M^N = [N|\underline{0}]M = [N|\overline{v}][\overline{v}|\underline{0}]M = [N|\overline{v}]M^{\overline{v}}$. Now, for this equality to hold we simply take v that is not in L and not used anywhere in N.
We then use typing preservation under term substitution, which leaves us with showing that $\Omega; (v : A@w') : \Gamma \vdash M^{\overline{v}} : B@w'$ – but that follows from the fact that $v \notin L$.
Proofs for two other cases that have changed compared to the proof described in Chapter 3, `unbox` $(\texttt{box}\, M) \mapsto_w M^w$ and `letdia` $[A]\texttt{get}\ w'\ (\texttt{here}\ M)N \mapsto_w (N^{w'})^M$, use the same idea of split substitution.

A.2.3 $\mathrm{L}_{\text{IS5-LF}}$ and $\mathrm{L}_{\text{IS5-Hyb}}$

Despite the fact that in $\mathrm{L}_{\text{IS5-LF}}$ we do not use worlds and in $\mathrm{L}_{\text{IS5-Hyb}}$ we do, these languages are similar enough that formalization and most of the proofs are almost identical, even in a proof assistant such as Coq. Most of the difficulties with both of these systems come from the background vs. current context split – with world-changing operations we need to control the exchange, but at the same time allow arbitrary order of contexts in the background (via two-level permutations, $G \cong G'$). To give an example look at a fragment of typing predicate for two versions of `unbox` in $\mathrm{L}_{\text{IS5-LF}}$:

```
Inductive types_LF : bg_LF -> ctx_LF -> te_LF -> ty -> Prop :=
(...)
| t_unbox_LF: forall G Gamma M A
  (Ok: ok_Bg_LF (Gamma :: G))
  (H: G |= Gamma |- M ::: [*] A),
  G |= Gamma |- unbox_LF M ::: A

| t_unbox_fetch_LF: forall G Gamma Gamma' M A
  (Ok: ok_Bg_LF (Gamma:: G & Gamma'))
  (H: G & Gamma' |= Gamma |- M ::: [*] A),
  forall G', G & Gamma ~=~ G' ->
    G' |= Gamma' |- unbox_LF M ::: A
(...)
where " G '|=' Gamma '|-' M ':::' A" := (types_LF G Gamma M A).
```

Here we explicitly use permutations of background contexts in order to express that in context-changing version of unbox operation, we can choose any context from the background.

Unfortunately, this leads to a lot of overhead, as if almost every proof we are forced to show that two environments are each other's permutation. We rely heavily on Coq's notion of setoids and morphisms to reduce that overhead as much as possible.

Another technical difficulty becomes visible when looking at lemmas on type preserving substitutions and merges. In Chapter 3 we have described worlds merging for $\mathrm{L}_{\text{IS5-L}}$; the formulation for $\mathrm{L}_{\text{IS5-Hyb}}$ would be the following:

If $G \vdash (w, \Gamma) \vdash M : A$ then $\{w_0|w_1\}((w, \Gamma) : G) \vdash \{w_0|w_1\}M : A$.

Here "$\{w_0|w_1\}((w, \Gamma) : G)$" merges contexts of w_0 and w_1. It then returns the after-merge equivalent of (w, Γ) as the current context and puts the rest to the background.
Note that there are three distinctive cases: $w_0 = w$, $w_1 = w$ or none of them are equal.

> In the first case, we add something from the background, enriching the current context (e.g. $\{w_0|w_1\}((w_0, \Gamma) : ((w_1, \Gamma') : G)) = ((w_0, \Gamma + \Gamma'), G)$).
>
> In the second, the current context changes its name and is enriched again: $\{w_0|w_1\}((w_1, \Gamma) : ((w_0, \Gamma') : G)) = ((w_0, \Gamma + \Gamma'), G)$.

In the last case, two contexts from the background are merged:
$\{w_0|w_1\}((w,\Gamma)\colon((w_0,\Gamma')\colon(w_1,\Gamma'')\colon G)) = ((w,\Gamma),(w_0,\Gamma'+\Gamma)\colon G)$.

This is already a lot of case analysis! But now consider that for term like `unbox-fetch` w' M we have to distinguish all these cases again, this time for w' instead of w.

Similar problems arise when dealing with type preservation under term substitution:

If $(w,\Gamma)\colon G \cong (w',\Gamma')\colon G'$, emptyEquiv $(G') \vdash w', \emptyset \vdash M\colon A$ and
$G^* \vdash (w,\Gamma)^* \vdash N\colon B$[2], then $G \vdash (w,\Gamma) \vdash [M|v]N\colon B$

Either $w = w'$ and we extend current context, or not – and additional assumption is added to one of the contexts in the background. Again, for context-changing terms like `unbox-fetch` w'' M, we have to consider cases $w'' = w'$ and $w'' \neq w'$.

Proofs of preservation and progress – except for requiring more case analysis and term rewriting – are no different than in $\mathrm{L}_{\text{IS5-L}}$ case.

A.3 Languages equivalence

Relations between languages described in Chapter 4 are also formalized in Coq. Conversions of terms usually turn out to be simple and straightforward – almost exactly like the pseudo-code we have shown in their descriptions. It is actually contexts rewriting that requires more work. When describing the translations we almost did not mention properties of rewritten contexts, as their correctness was intuitive and matched our expectations (e.g. bucket-sorting a context does not change its correctness with respect to variable repetition). In Coq we have to formalize all these properties. Additionally, we now want the translations to preserve locally closed properties – this is something we did not have to consider previously.

Rewriting contexts and terms for translations from $\mathrm{L}_{\text{IS5-Hyb}}$ to $\mathrm{L}_{\text{IS5-L}}$ and to $\mathrm{L}_{\text{IS5-LF}}$ is very simple; Coq implementation of these rewrites is exactly as described in Chapter 4. The proofs are also straightforward and easy to follow, allowing us to skip description of these translations and focus on the two remaining cases.

[2] $(w,\Gamma)^*$ means that if $w = w'$ then we extend context with a pair (v, A); G^* does that for every context in the background

A.3.1 From $L_{IS5\text{-}L}$ to $L_{IS5\text{-}Hyb}$

When describing this translation we have mentioned that rewriting contexts between $L_{IS5\text{-}L}$ and $L_{IS5\text{-}Hyb}$ is done via bucket-sort procedure. This proves to be particularly tricky to formalize – even more so to prove correctness.

```
Fixpoint gather_keys_L (k: var) (l: ctx_L) :=
match l with
| nil => nil
| (k', v) :: l' =>
  if (eq_var_dec k k')
  then v :: (gather_keys_L k l')
  else gather_keys_L k l'
end.

Fixpoint bucket_sort_L
         (keys: worlds_L)
         (l: ctx_L) :=
match keys with
| nil => nil
| k :: keys' =>
  (k, gather_keys_L k l) :: bucket_sort_L keys' l
end.
```

So far there is nothing worrying here – for each world k we gather all assumptions regarding k from l. The problematic part begins when we want to obtain the result as a pair (background, current context). We cannot be sure that the world marked as current in $L_{IS5\text{-}L}$ is actually in Ω – therefore we have to make the current context optional:

```
Fixpoint split_at_Hyb (l: bg_Hyb) (k: var) :=
match l with
| nil => (nil, None)
| (k', l) :: l' =>
  if (eq_var_dec k k')
    then (l', Some (k', l))
  else
    let res := split_at_Hyb l' k in
    ((k',l) :: fst res, snd res)
end.
```

```
Definition L_to_Hyb_ctx
         (Omega_L: worlds_L)
         (Gamma_L: ctx_L)
         (w_L: var) :
         (bg_Hyb * option ctx_Hyb) :=
  let G := bucket_sort_L Omega_L Gamma_L in
  split_at_Hyb G w_L.
```

In order for this procedure to be useful we have to show that if $w_{\mathrm{L}} \in \Omega$ then the result is not None.

When using permutations, in many lemmas for $\mathrm{L}_{\text{IS5-Hyb}}$ (and $\mathrm{L}_{\text{IS5-LF}}$) we do not care about the distinction between current context and background – it is therefore useful to realize, that bucket-sort result is in fact a permutation LtoHybCtx result. This will be used in the proof that a correct (Ok_{L}) context from $\mathrm{L}_{\text{IS5-L}}$ rewrites to a correct one from $\mathrm{L}_{\text{IS5-Hyb}}$ ($\mathrm{Ok}_{\mathrm{Hyb}}$).

As to the rewrite of terms – there are two things to notice, highlighted also in Chapter 4. First is that in $\mathrm{L}_{\text{IS5-L}}$, in terms like `unbox` M or `here` M, there is no mention of the current world within them. To rewrite them into $\mathrm{L}_{\text{IS5-Hyb}}$ without the knowledge of current world is impossible – thus, our function takes current world (be it bound or free) as an argument, resulting in the following signature:

```
Fixpoint L_to_Hyb_term (w: vwo) (M0: te_L) : te_Hyb
```

Second, as $\mathrm{L}_{\text{IS5-Hyb}}$ does not have `fetch` or `get` operations, we need to simulate them somehow. Back in Chapter 4 we said that for technical reasons we want to rewrite `fetch` w' M in world w into an overly complicated term:

$\texttt{fetch}_{\mathrm{L}}\ w'\ M \Rightarrow$
let $w_0 :=$ freshWorld in
  let $w'' :=$ freshWorld in
  let $v :=$ freshVar in
  let $N := \texttt{box}_{\mathrm{Hyb}}\ w_0\ (\texttt{unbox-fetch}\ \ w''(\texttt{hyp}\, vA))$ in
  $\texttt{letdia-get}_{\mathrm{Hyb}}\ w'\ [v_A := w'']\ \texttt{get-here}_{\mathrm{Hyb}}\ w'\ (\mathrm{LtoHyb}(M, w'))N$

In current term representation this does not require generation of fresh variables:

$\texttt{fetch}_{\mathrm{L}}\ w'\ M \Rightarrow$
let $N := \texttt{box}_{\mathrm{Hyb}}\,(\texttt{unbox-fetch}\ \ \underline{0}(\texttt{hyp}\,\underline{1}A))$ in
$\texttt{letdia-get}_{\mathrm{Hyb}}\,{}^{A}w'[\texttt{get-here}_{\mathrm{Hyb}}\ w'\ (\mathrm{LtoHyb}(M, w'))]N,$

but this is still not an obvious rewrite to propose. The most natural way to rewrite $\texttt{fetch}\ w\ M$ is by using $\texttt{box}\,(\texttt{unbox-fetch}\ w\ (\mathrm{LtoHyb}(M, w)))$. What is the problem with such approach? Let us take a closer look.

If there are no bound world variables[3] in $\texttt{fetch}\ w'\ M$, then indeed $\mathrm{LtoHyb}(\texttt{fetch}\ w'\ M, w) = \texttt{box}\,(\texttt{unbox-fetch}\ w\ (\mathrm{LtoHyb}(M, w)))$. Otherwise we need to take into account that box adds one bound world variable – simplest example to observe how such a rewrite works is to take $w' = \underline{0}$. Then $\mathrm{LtoHyb}(\texttt{fetch}\ \underline{0}\ M, w) = \texttt{box}\,(\texttt{unbox-fetch}\ \underline{0}\ \mathrm{LtoHyb}(M, w'))$. This does not look right - we started with a term in $\mathrm{L_{IS5\text{-}L}}$ that was not locally closed, but ended up with one that is!
We can of course try to fix it by shifting: $\mathrm{LtoHyb}(\texttt{fetch}\ w'\ M, w) \Rightarrow \texttt{box}\,(\texttt{unbox-fetch}\ w'^{\uparrow}\ (\mathrm{LtoHyb}(M, w')))$, but the same problem may re-appear in the result of $\mathrm{LtoHyb}(M, w')$.
Another solution is to actually shift the resulting term[4] – that is:
$\mathrm{LtoHyb}(\texttt{fetch}\ w'\ M, w) = \texttt{box}\,(\texttt{unbox-fetch}\ w'^{\uparrow}\ (\mathrm{LtoHyb}(M, w'))^{\uparrow})$. This is better, but very unnatural to work with – for example, how do we tell when a shifted term types?

Our solution is to ignore this problem for as long as possible by postponing the introduction of new world variable (in box). Note that in

$$\begin{aligned}&\mathrm{HybtoL}\,(\texttt{fetch}_{\mathrm{L}}\ w'\ M)w = \\ &\text{let } N := \texttt{box}_{\mathrm{Hyb}}\,(\texttt{unbox-fetch}\ \ \underline{0}(\texttt{hyp}\,\underline{1})A) \text{ in} \\ &\texttt{letdia-get}_{\mathrm{Hyb}}\ {}^{A}w'[\texttt{get-here}_{\mathrm{Hyb}}\ w'\ (\mathrm{LtoHyb}(M, w'))]N,\end{aligned}$$

we do not create any new world variables for the subterm in which we make a recursive call.

Complete term rewriting function is as follows:

```
Fixpoint L_to_Hyb_term (w: vwo) (M0: te_L) : te_Hyb :=
match M0 with
| hyp_L v => hyp_Hyb v
| lam_L A M => lam_Hyb A (L_to_Hyb_term w M)
| appl_L M1 M2 =>
      appl_Hyb (L_to_Hyb_term w M1) (L_to_Hyb_term w M2)
| box_L M => box_Hyb (L_to_Hyb_term (bwo 0) M)
| unbox_L M =>  unbox_fetch_Hyb w (L_to_Hyb_term w M)
| here_L M => get_here_Hyb w (L_to_Hyb_term w M)
```

[3]Note that this is a stronger requirement than $\mathrm{LC_w}\,M$

[4]We define such operation inductively as a shift on each world in the term

```
| letd_L M1 M2 =>
      letdia_get_Hyb w
           (L_to_Hyb_term w M1)
           (L_to_Hyb_term (shift_vwo w) M2)
| fetch_L w' M =>
      letdia_get_Hyb w'
           (get_here_Hyb w' (L_to_Hyb_term w' M))
           (box_Hyb (unbox_fetch_Hyb (bwo 1) (hyp_Hyb (bte 0))))
| get_L w' M =>
      letdia_get_Hyb w'
          (L_to_Hyb_term w' M)
          (get_here_Hyb (bwo 0) (hyp_Hyb (bte 0)))
end.
```

Note that in the `box` case recursion call we use a bound world variable instead of a fresh one. This is fine because we open this term with a fresh world in the premise of typing rule for `box`.

Proof of typing preservation is again simple, but with a lot of overhead coming from context rewriting.

We also provide an alternative translation using a relation. It is defined based on the function described above, except it connects $\texttt{unbox}_{\text{L}}\ M$ with every $\texttt{unbox-fetch}_{\text{Hyb}}\ w\ M'$ for which M is connected to M' (same for `here` and `letdia`). Obviously this relation is capturing too many pairs or terms, but – as it does not use world as a parameter – it is much simpler to work with. In particular, we can prove lemma regarding term substitution without requiring any term to be stable.

```
Lemma L_to_Hyb_term_R_subst_t:
forall M M' C1 C2 v,
  L_to_Hyb_term_R C1 C2 ->
  L_to_Hyb_term_R M M' ->
  L_to_Hyb_term_R (subst_t_L C1 v M) (subst_t_Hyb C2 v M').
```

It is trivial to see that pair $(M, \text{LtoHyb}(M, w))$ will satisfy this relation. We can then show the value preservation lemma for relation and use this result for a function:

```
Lemma L_to_Hyb_term_R_value:
forall M M',
 lc_w_Hyb M' -> lc_t_Hyb M' ->
```

```
  value_L M -> L_to_Hyb_term_R M M' ->
  value_Hyb M' \/
  forall w, exists M'', steps_Hyb (M', w) (M'', w) /\ value_Hyb M''.

Lemma L_to_Hyb_term_value:
forall M M' w,
 lc_w_Hyb M' -> lc_t_Hyb M' ->
  value_L M -> (M' = L_to_Hyb_term w M) ->
  value_Hyb M' \/
  exists M'', steps_Hyb (M', w) (M'', w) /\ value_Hyb M''.
intros; destruct L_to_Hyb_term_R_value with M M'; subst; eauto.
Qed.
```

As the relation does not need worlds, the proof of value preservation for it is almost trivial.

A.3.2 From $L_{IS5\text{-}LF}$ to $L_{IS5\text{-}Hyb}$

In the second translation we will describe here, we cannot use functions to rewrite neither contexts nor terms. For contexts, our motivation is mostly pragmatic – a function assigning fresh names to contexts would most probably be order-dependent. It is very likely that we will need permutations in some of the proofs, therefore it is preferable to define a relation accepting any unique assignment of names to contexts. The relation we define is simply:

```
Definition LF_to_Hyb_ctx (G: bg_LF) (G': bg_Hyb) :=
  PPermut_LF (map snd_ G') G /\ ok_Hyb G' nil.
```

In case of terms, we do not have much of a choice. This transformation can only work for well-typed terms – and we do actually use type trees to create resulting term in $L_{IS5\text{-}Hyb}$. In a way, this development is formalizing the table we have shown in chapter 4.

Signature of the relation is the following:

```
Inductive LF_to_Hyb_term_R: ctx_LF  -> te_LF  -> ty ->
                            bg_Hyb -> var -> te_Hyb -> Prop
```

In each case, the relation will require that the term from $L_{IS5\text{-}LF}$ types in the environment passed as one of the arguments. If that is the case, we can reconstruct the corresponding $L_{IS5\text{-}Hyb}$ term.

Note that we use only the environment from $L_{IS5\text{-}Hyb}$ – this is to provide consistency with the induction hypothesis. For example in case of unbox:

```
| unbox_LF_Hyb:
    forall G Gamma G' w M N A,
      G |= Gamma |- (unbox_LF M) ::: A ->
      LF_to_Hyb_ctx (Gamma::G) ((w, Gamma)::G') ->
      LF_to_Hyb_term_R Gamma M ([*]A) G' w N ->
      LF_to_Hyb_term_R Gamma (unbox_LF M) A
                    G' w (unbox_fetch_Hyb (fwo w) N)

| unbox_fetch_LF_Hyb:
    forall G Gamma Gamma' G' w w' M N A G'',
      w <> w' ->
      (Gamma'::G) |= Gamma |- (unbox_LF M)  ::: A ->
      LF_to_Hyb_ctx (Gamma::Gamma'::G)
                    ((w, Gamma)::(w', Gamma')::G') ->
      LF_to_Hyb_term_R Gamma' M ([*]A)
                    ((w, Gamma)::G') w' N ->
      ((w', Gamma')::G') ~=~ G'' ->
      LF_to_Hyb_term_R Gamma (unbox_LF M) A
                    G'' w (unbox_fetch_Hyb (fwo w') N)
```

we need to be certain that there was no change in assignment of worlds to contexts. Otherwise we will not be able to distinguish two above cases.

It is not a surprise that a relation using typing preserves types. The preservation of values is also trivial to show.

A.4 Termination of evaluation

Termination of evaluation is particularly interesting to formalize in Coq because from it we can extract the evaluator. This requires the definitions to be in sort Type rather than Prop, but this is usually easy to change – in most cases with minimal effort. In our formalization we give a proof of $L_{IS5\text{-}LF}$ without $\Diamond$ in an extractable form, leaving the rest of languages formalized with proof-irrelevance.

This is a technical decision. We normally treat permutations as equivalence relations with all the benefits of rewriting using Setoid type. When moving from Prop to Type it turns out to be necessary to change the definition of permutation ($\cong$) as well – and this of course disallows us from

treating permutations as relations[5]. Such change is surely not critical – we can still prove everything we could before – but it requires all rewrites to be more manual, as we cannot use setoid rewrites.

It is worth noting that these technical difficulties are most probably due to ambiguity of typing. When using a la Church type system with disambiguous typing, changes going so deeply would most likely be avoidable.

We should also mention that to prove termination for a sublanguage without $\Diamond$, we actually have to re-create such language with type $\Diamond$ and terms that use it (`here`, `letdia`, etc.) removed – it is a bit cumbersome, but otherwise analysis of $\Diamond$ type has to be included in the proof.

In the next part of this section we intend to describe the $\Diamond$-free sublanguage of $\mathrm{L}_{\text{IS5-LF}}$. Next we turn our attention to the complete $\mathrm{L}_{\text{IS5-LF}}$ laguage.

A.4.1 $\mathrm{L}_{\text{IS5-LF}}$ without $\Diamond$

The definitions of properties and lemmas formalized in Coq differ from the ones described in the thesis only in that the former have to ensure that terms are locally closed and contexts correct. Therefore, the definitions of termination and reducibility in $\mathrm{L}_{\text{IS5-LF}}$ are as follows:

```
Inductive WHT: te_LF -> Type :=
| val_WHT: forall M, value_LF M -> WHT M
| step_WHT: forall M,
              { V & (value_LF V) * (steps_LF M V)) -> WHT M.


Fixpoint Red (M: te_LF) (A: ty) : Type :=
match A with
| tvar => WHT M
| tarrow A1 A2 =>
    (WHT M) *
    (forall N,
       lc_t_LF N ->
       Red N A1 ->
      Red (appl_LF M N) A2)
| tbox A1 => (WHT M) * (Red (unbox_LF M) A1)
end.
```

[5]This is because in Coq relation is a defined as $A \to A \to \mathrm{Prop}$

As this sublanguage has the property that every term has at most one redex, we can show that

```
Lemma WHT_step:
forall M M',
  WHT M ->
  M |-> M' ->
  WHT M'.

Lemma WHT_step_back:
forall M M',
  M |-> M' ->
WHT M' -> WHT M.
```

Next, we can define and prove the three properties (CR1 – CR3) that we require to hold for reducibility relation; these proofs are completely straightforward, we will therefore omit their descriptions. They state that reduction preserves reducibility (in both directions) and that every reducible term's evaluation terminates.

With these definitions and properties ready, the next step is defining simultaneous substitution.

```
Fixpoint SL (L: list (var * ty * te_LF)) (M: te_LF) : te_LF :=
match M with
| hyp_LF (bte v) => M
| hyp_LF (fte v) =>
  let x := find_var L v in
  match x with
    | Some (v, A, M) => M
    | None => hyp_LF (fte v)
  end
| lam_LF A M => lam_LF A (SL L M)
| appl_LF M N => appl_LF (SL L M) (SL L N)
| box_LF M => box_LF (SL L M)
| unbox_LF M => unbox_LF (SL L M)
end.
```

When describing the properties of this substitution ($[L|X]N$) in Chapter 5, we have mentioned that the list of variables X should contain all free variables from term M. For this concrete implementation, we might want to be more specific. There is a number of options:

1. We may require exactly variables from $\mathrm{FV}(M)$ – this causes a number of problems for example in (othewise trivial) $M \cdot N$ case, as free variables from M and N do not have to be exactly the same;

2. We may require at least the variables from $\mathrm{FV}(M)$ to be present, which solves the above problem;

3. We can also simply take all the known variables, that is every variable present in the environment.

The second variant will require an additional lemma stating that "a term well-typed in an empty environment has no free term variables" to obtain the termination result; the third one makes transition between reducibility theorem and termination theorem the simplest. We therefore chose to use all known term variables in the substitution.

The complete formalization of reducibility theorem in $\mathrm{L}_{\mathrm{IS5\text{-}Hyb}}$ is therefore the following:

```
Theorem red_theorem:
forall G Gamma M A,
  lc_t_LF M ->
  G |= Gamma |- M ::: A ->
  forall L,
    OkL L nil ->
    concat(Gamma::G) *=* map fst_ L ->
    (forall a b c, Mem (a,b,c) L -> lc_t_LF c) ->
    (forall a b c, Mem (a,b,c) L -> Red c b) ->
    Red (SL L M) A.
```

There are three cases worth mentioning: `hyp`, λ and `box`.

– In `hyp` $\overline{v}A$ case (with $(v, A) \in \Gamma$) we have to show that the substitution was indeed done – that is, by the definition, that we have found a tuple containing variable v in L. Then the term substituted for `hyp` $\overline{v}$A is, by our assumptions on L, reducible. It remains to argue that it is in fact reducible exactly at A.

– For $\lambda A.M$ we have to show that for every reducible term N, it is the case that $\mathrm{Red}_B\,((\lambda A.(\mathrm{SL}\,L\ W\ M)) \cdot N)$. By using one of the properties of reducibility (CR3), we end up having to prove:

```
n : x \notin (...)
X1 : forall V : te_LF,
     Red V A ->
     lc_t_LF V ->

     Red (SL ((x, A, V) :: L0)
     [hyp_LF (fte x) // bte 0]M) B
N : te_LF
H3 : lc_t_LF N
X2 : Red N A
============================
 Red (SL L0 M ^t^ N) B
```

We will use the same split of substitution as we have described for the Preservation proof in Section 1: $(\mathrm{SL}\, L_0\ W\ M)^N = [N|x](\mathrm{SL}\, L_0\ W\ M)^x$. Now we can extend L_0 with (x, A, V) and change $(\mathrm{SL}\, L_0\ W\ M)^x$ into $\mathrm{SL}\, L_0\ W\ M^x$, as all terms in L_0 are locally closed – therefore opening does not affect them.

– Lastly, `box` M follows the same pattern as $\lambda A.M$, except extending W rather than L.

Finally, the termination theorem is simply given as:

```
Theorem termination_theorem:
forall G M A,
  emptyEquiv_LF G |= nil |- M ::: A ->
  WHT M.
```

A.4.2 Complete $\mathbf{L_{IS5\text{-}LF}}$

In the description of termination for the complete $\mathrm{L_{IS5\text{-}LF}}$ language we have introduced some new concepts, not present in the sublanguage variant. These include continuations and mutually inductive logical relations.

First, as conceptually complicated as they are, continuations are actually almost trivial to formalize:

```
Inductive Cont :=
| IdK: ty -> Cont
| ConsU: Cont -> ty -> Cont
| ConsD: Cont -> te_LF -> ty -> ty -> Cont
| ConsA: Cont -> te_LF -> ty -> ty -> Cont.
```

```
Fixpoint ContAppl (K: Cont) (M: te_LF) : te_LF :=
match K with
| IdK A => M
| ConsU K' A => ContAppl K' (unbox_LF M)
| ConsD K' N A B => ContAppl K' (letdia_LF M N)
| ConsA K' N A B => ContAppl K' (appl_LF M N)
end.

Notation " K '@' M " := (ContAppl K M) (at level 65).

Inductive ContLC: Cont -> Prop :=
| ContId_lc: forall A, ContLC (IdK A)
| ContU_lc: forall K A, ContLC K -> ContLC (ConsU K A)
| ContD_lc: forall K N A B,
                ContLC K -> lc_t_n_LF 1 N ->
                ContLC (ConsD K N A B)
| ContA_lc: forall K N A B,
                ContLC K -> lc_t_LF N ->
                ContLC (ConsA K N A B)
```

Logical relations for $L_{IS5\text{-}LF}$ on the other hand, however rather straightforward and simple to describe on paper, are actually quite tricky[6] to encode in Coq. There are two reasons for this – first is that we cannot define these relations as inductive types (just as we couldn't do it in the sublanguage), since Coq requires strictly positive occurrences of inductively defined types in their definitions. Second, when using fixpoint (or function) with mutually recursive definitions, the system requires such recursion to be structural – meaning that we have to explicitly use "smaller" argument for the recursion call.

[6] And mostly unreadable.

This leads to the following definition of logical relations:

```
Fixpoint R M A {struct A} : Prop :=
let Q M A := forall K,
               lc_t_LF M -> ContLC K -> RC K A ->
               WT (K @ M) in
match A with
| tvar => WT M
| A1 ---> A2 => WT M /\
                (forall N,
                   lc_t_LF N ->
                   Q N A1 ->
                   Q (appl_LF M N) A2)
| [*]A1 => WT M /\ forall K, RC K A1 -> WT (K @ (unbox_LF M))
| <*>A1 => forall K, ContLC K ->
                     (forall V,
                        lc_t_LF V ->
                        R V A1 ->
                        WT (K @ (here_LF V))) ->
                     WT (K @ M)
end
with RC K A {struct A} : Prop :=
let Q M A := (..) (* same as above *)
match A with
| tvar => forall V, lc_t_LF V -> WT V -> WT (K @ V)
| tbox A1 => forall V, lc_t_LF V ->
                       (WT V /\
                        forall K0, ContLC K0 ->
                                   RC K0 A1 ->
                                   WT (K0 @ (unbox_LF V))) ->
                       WT (K @ V)
| A1 ---> A2 => forall V, lc_t_LF V ->
                  (WT V /\
                   (forall N,
                      lc_t_LF N ->
                      Q N A1 ->
                      Q (appl_LF V N) A2)) ->
                  WT (K @ V)
| <*> A1 => forall V, lc_t_LF V ->
                       Q V A1 -> WT (K @ (here_LF V))
end.
```

Luckily, the proof of reducibility is straightforward enough that complex analysis of these two relations is not required.

```
Theorem red_theorem:
forall G Gamma M A,
  lc_t_LF M ->
  G |= Gamma |- M ::: A ->
 forall L,
    OkL L nil ->
    concat(Gamma::G) *=* map fst_ L ->
    (forall a b c, Mem (a,b,c) L -> lc_t_LF c) ->
    (forall a b c, Mem (a,b,c) L -> Q c b) ->
    forall K,
      ContLC K ->
      RC K A ->
      WT (K @ (SL L M)).
```

Main result – the termination theorem – is defined exactly as in the sublanguage case.

A.4.3 Evaluator extraction

One final piece of code that deserves mention in the technical part of this thesis is the evaluator extracted from termination proof for ${}^{\setminus\Diamond}L_{\text{IS5-LF}}$. The code that – upon running – produces the (almost) complete evaluator can be as simple as:

```
Extraction "termination_LF_nodia" termination_language.
```

If however we feel that there is no need to re-define types such as list, nat, bool etc – we can add the following lines:

```
Extract Inductive bool => "bool" [ "true" "false" ].
Extract Inductive list => "list" [ "[]" "(::)" ].
Extract Inductive prod => "(*)"  [ "(,)" ].
Extract Inductive sumbool => "bool" [ "true" "false" ].

Extract Inductive nat => "int"
  [ "0" "(fun x -> x + 1)" ]
  "(fun zero succ n ->
      if n=0 then zero () else succ (n-1))".
```

This is completely optional. What is not is the realization of axioms. When extracting code with unrealized axioms, we are being warned:

```
Warning: The following axioms must be realized in the
extracted code:
  eq_var_dec LibAxioms.indefinite_description.
```

In case of the evaluator we have extracted, the two axioms both have to do with variables – decidability of their equality and creation of a fresh one.

Looking into the extracted code (or an external Coq library that produces it – `tlc`) makes us realize that var is defined as interger, so `eq_var_dec` can be realized using equality for integers.

As for the other axiom – it comes from the same `tlc` library and is used only in the definition of fresh variable generator taking a finite set as an argument.

```
(** val indefinite_description : __ -> 'a1 **)

let indefinite_description =
  failwith "AXIOM TO BE REALIZED"

(** val classicT : bool **)

let classicT =
  let h = indefinite_description __ in if h then true else false
(...)
(** val epsilon_def : __ -> 'a1 **)

let epsilon_def _ =
  if classicT then indefinite_description __ else inhab_witness __

(** val epsilon : __ -> 'a1 **)

let epsilon _ =
  epsilon_def __

module Variables =
 struct
  type var = int
(...)
  type vars = var FsetImpl.fset

  (** val var_gen_list : int list -> int **)

  let var_gen_list l =
```

```
    plus ((fun x -> x + 1) 0) (fold_right plus 0 l)

  (** val var_gen : vars -> var **)

  let var_gen e =
    var_gen_list (epsilon __)
(...)
```

In this case we might want to replace the entire finite set module with a standard set implementation. Then, we can use standard function `elements` to move from `var_gen` to `var_gen_list`.

After making these two changes we have a complete evaluator of ${}^{\setminus\Diamond}L_{\text{IS5-LF}}$ extracted from the proof.

Bibliography

[1] Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In George C. Necula and Philip Wadler, editors, *Proceeding of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 3–15. ACM, 2008.

[2] Roberta Ballarin. Modern origins of modal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2010 edition, 2010.

[3] Małgorzata Biernacka and Dariusz Biernacki. A context-based approach to proving termination of evaluation. *Electron. Notes Theor. Comput. Sci.*, 249:169–192, August 2009.

[4] Arthur Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, pages 1–46, 2011. 10.1007/s10817-011-9225-2.

[5] Thierry Coquand and Gerard Huet. The calculus of constructions. *Inf. Comput.*, 76(2-3):95–120, February 1988.

[6] Frederic Brenton Fitch. *Symbolic logic : An introduction.* The Ronald Press Company, Cop., New York, 1952.

[7] Didier Galmiche and Yakoub Salhi. Label-free proof systems for intuitionistic modal logic is5. In *Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 255–271, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, New York, NY, USA, 1989.

[9] Simo Knuuttila. Medieval theories of modality. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2013 edition, 2013.

[10] Saul Kripke. A completeness theorem in modal logic. *J. Symb. Log.*, 24(1):1–14, 1959.

[11] Sam Lindley and Ian Stark. Reducibility and $\top\top$-lifting for computation types. In *Proceedings of the 7th international conference on Typed Lambda Calculi and Applications*, TLCA'05, pages 262–277, Berlin, Heidelberg, 2005. Springer-Verlag.

[12] Tom Murphy, VII, Karl Crary, and Robert Harper. Type-safe distributed programming with ML5. In *Trustworthy Global Computing 2007*, November 2007.

[13] Tom Murphy VII. *Modal types for mobile code*. PhD thesis, Pittsburgh, PA, USA, 2008. AAI3314655.

[14] Tom Murphy VII, Karl Crary, Robert Harper, and Frank Pfenning. A symmetric modal lambda calculus for distributed computing. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, LICS '04, pages 286–295, Washington, DC, USA, 2004. IEEE Computer Society.

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

[16] Gordon D. Plotkin and Colin Stirling. A framework for intuitionistic modal logics. In Joseph Y. Halpern, editor, *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, March 1986*, pages 399–406. Morgan Kaufmann, 1986.

[17] A. N. Prior. *Time and Modality*. Greenwood Press, 1955.

[18] Alex K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.