UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Typechecking Contextual Modal Programs

MASTER'S THESIS

Student: **Kevin Alexander López Aquino**

Advisor: **Aleksandar Nanevski** (IMDEA Software Institute)

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Typechecking Contextual Modal Programs

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF:
Máster en Métodos Formales en Ingeniería Informática

Author: **Kevin Alexander López Aquino**

Advisor: **Aleksandar Nanevski** (IMDEA Software Institute)

September 2025

Abstract

This master's thesis studies Contextual Modal Type Theory (CMTT), a modal type theory in which the standard necessity and possibility modalities are relativized to variable contexts. We present CMTT in a bidirectional style, enabling a straightforward implementation of a typechecker, and give operational semantics for executing contextual modal programs. Our development begins with the simply-typed lambda calculus and gradually enriches it with new constructs, making a stop along the way to study the modal type theory of Pfenning and Davies. Throughout, we examine both the computational and logical aspects of the systems presented under the lens of the Curry–Howard correspondence.

Keywords: *modal logic, modal types, Curry-Howard correspondence, bidirectional typechecking, contextual modalities, operational semantics.*

Resumen

Esta trabajo fin de máster estudia la Contextual Modal Type Theory (CMTT), una teoría de tipos modales en la que las modalidades usuales de necesidad y posibilidad se relativizan a contextos de variables. Presentamos CMTT en un estilo bidireccional, lo que permite una implementación directa de un *typechecker*, y damos una semántica operacional para la ejecución de programas modales contextuales. Nuestro desarrollo parte del cálculo lambda simplemente tipado y lo enriquece gradualmente, haciendo una pausa en el camino para estudiar la teoría de tipos modal de Pfenning y Davies. A lo largo del trabajo, examinamos tanto los aspectos computacionales como los lógicos de los sistemas presentados, bajo la óptica de la correspondencia de Curry–Howard.

Palabras clave: *lógica modal, tipos modales, correspondencia de Curry-Howard, tipado bidireccional, modalidades contextuales, semántica operacional.*

Acknowledgments

I would like to thank Aleks for his intellectual generosity. He first welcomed me as an intern, and since then has always been open to discussing my doubts, however minor, with patience and care. His hands-on approach and ability to pose the right challenges have been invaluable in guiding my work. I have greatly appreciated his commitment to rigor and his understanding as my research interests occasionally shifted from one topic to another—much like a butterfly drawn to a new flower.

I would also like to thank everyone at the IMDEA Software Institute for creating a stimulating and welcoming environment, and for the many conversations, suggestions, and small acts of help that made this work possible. Finally, I am deeply grateful to my family for their unwavering support, encouragement, and belief in me throughout this wonderful, challenging, and sometimes crazy journey.

Contents

List of Figures

List of Code Snippets

0 Introduction

Contextual Modal Type Theory (CMTT) (Nanevski, Pfenning, and Pientka, 2008) is the type-theoretic counterpart to intuitionistic contextual modal logic, a system that extends standard modal logic by relativizing the modalities of necessity ($\Box$) and possibility ($\Diamond$) to explicit contexts of propositions.

From a computational perspective, one way to think about modal type theory is in terms of code manipulation. The modality $\Box$ corresponds to *closed source code*—code that may contain metavariables but no free program variables—and the constructor **box** e of type $\Box A$ acts as a quotation operator, akin to `quote` in Lisp. This enables programs to treat code as data while preserving type safety. The type $\Box A \to A$, for instance, reflects the idea that, given closed source code of type A, we can compile it to obtain executable code of type A. CMTT refines this perspective by supporting *open source code*—code that depends on a specific context of assumptions. This makes modal logic a powerful framework for expressing how programs interact with their surrounding context, with applications to computational effects and metaprogramming.

In this work, we present a method for typechecking CMTT programs based on *bidirectional typechecking*, a well-established technique that separates typing judgments into checking and synthesis modes. Once the rules are formulated in this style, implementing a typechecker becomes straightforward. With a practical typechecking method in place, we then define an operational semantics and implement an interpreter, which enables the execution of well-typed CMTT programs.

To reach this goal, we proceed incrementally, beginning with the foundational simply-typed lambda calculus and gradually introducing the necessary logical and computational constructs that culminate in the full CMTT system.

We now outline the structure of the thesis.

Chapter 1 establishes the foundational background for the rest of the work. Section 1.1 begins with a discussion of key philosophical notions—most notably, the distinction between judgments and propositions, as analyzed by Martin-Löf. We then introduce intuitionistic propositional logic in Section 1.2 using the system of natural deduction. For the implicational fragment of propositional logic, the rules are as follows:

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

$$\frac{\Gamma, x{:}A\ \mathtt{true} \vdash B\ \mathtt{true}}{\Gamma \vdash A \to B\ \mathtt{true}}\ {\to}\mathtt{I}^x \qquad \frac{\Gamma \vdash A \to B\ \mathtt{true} \qquad \Gamma \vdash A\ \mathtt{true}}{\Gamma \vdash B\ \mathtt{true}}\ {\to}\mathtt{E}$$

Using these rules, we can construct derivation trees such as the one shown below, which constitutes evidence for the judgment $\vdash A \to B \to A\ \mathtt{true}$.

$$\dfrac{\dfrac{\dfrac{}{x{:}A\ \mathtt{true}, y{:}B\ \mathtt{true} \vdash A\ \mathtt{true}}\ \mathtt{hyp}_x}{x{:}A\ \mathtt{true} \vdash B \to A\ \mathtt{true}}\ {\to}\mathtt{I}^y}{\vdash A \to B \to A\ \mathtt{true}}\ {\to}\mathtt{I}^x$$

In this way, we say that the judgment $\vdash A \to B \to A\ \mathtt{true}$ is *synthetic*, since its evidence must be constructed explicitly. In contrast, there are also *analytic* judgments, whose evidence is already contained in the terms of the judgment itself. We can convert synthetic judgments as the one shown before into analytic ones by introducing annotations that indicate the inference rules to be applied when reconstructing a derivation. As annotations, we use terms from the lambda calculus, defined by the following grammar:

$$e ::= x \mid \lambda x.\, e \mid e_1\ e_2$$

The inference rules can now be annotated as follows, with propositions marked in blue and annotations in green:

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

$$\frac{\Gamma, x{:}A \vdash e : B}{\Gamma \vdash \lambda x.\, e : A \to B}\ {\to}\mathtt{I}^x \qquad \frac{\Gamma \vdash e_1 : A \to B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\ e_2 : B}\ {\to}\mathtt{E}$$

Let us return now to the derivation tree for the judgment $\vdash A \to B \to A\ \mathtt{true}$ we saw earlier. Once annotated with terms, we obtain:

$$\dfrac{\dfrac{\dfrac{}{x{:}A, y{:}B \vdash x{:}A}\ \mathtt{hyp}_x}{x{:}A \vdash \lambda y.\, x : B \to A}\ {\to}\mathtt{I}^y}{\vdash \lambda x.\, \lambda y.\, x : A \to B \to A}\ {\to}\mathtt{I}^x$$

Observe that the full structure of the derivation is now encoded in the final judgment of the tree:

$$\vdash \lambda x.\, \lambda y.\, x : A \to B \to A$$

This compact representation makes the judgment evident from its terms alone, without the need to reconstruct the full derivation tree. In this sense, we have obtained an analytic judgment (Martin-Löf, 1994).

The annotations introduced here not only simplify proofs in natural deduction, but also constitute the core of the Curry–Howard correspondence, which we explore in Section 1.3. According to this correspondence, propositions are interpreted as *types*, and proof terms as *programs*. Thus, a judgment like

$$\vdash \lambda x.\, \lambda y.\, x : A \to B \to A$$

not only expresses that the proposition $A \to B \to A$ is true by providing a method for proving it, but also corresponds to a well-typed program in a functional programming language. For example, our rule for modus ponens, $\to \mathtt{E}$, specifies the conditions under which function application is valid.

Another important aspect of the Curry–Howard correspondence arises when considering the β-reduction and η-expansion rules of the lambda calculus. For the implicational fragment, these are:

$$\begin{array}{lll} \Gamma \vdash (\lambda x.\, e)\, e' : B & \Longrightarrow_R & \Gamma \vdash [e'/x]e : B \\ \Gamma \vdash e : A \to B & \Longrightarrow_E & \Gamma \vdash \lambda x.\, e\ x : A \to B \end{array}$$

Analyzing their logical content brings us to the notion of *harmony*. This concept expresses that the introduction and elimination rules associated with a logical connective must not be chosen arbitrarily—they must be in balance, such that neither goes beyond nor falls short of the other. Several formulations of harmony exist (Francez and Dyckhoff, 2012), but one especially relevant in the context of computation requires both *local soundness* and *local completeness* (Pfenning and Davies, 2001; Pfenning, 2009).

Local soundness ensures that elimination rules do not yield more than what is justified by their corresponding introduction rules. This is shown by demonstrating that any derivation containing a *detour*—an elimination rule applied immediately after an introduction rule—can be replaced by a more direct derivation of the same conclusion without the detour.

Local completeness, on the other hand, guarantees that elimination rules are not too weak with respect to the introduction rules. To demonstrate this, we show that there is a way to apply the elimination rules so that we can reconstruct a proof of the original conclusion.

Beyond revealing a deep connection between logic and computation, the Curry–Howard correspondence serves as a guiding principle in the design of programming languages grounded in logic. In this context, types are not merely labels—they function as specifications that describe what a program must satisfy. At the same time, types act as abstractions that hide away implementation details while ensuring the preservation of essential properties. This logical foundation enables rigorous reasoning about programs: a well-typed program inherently carries a proof of its own correctness and reliability.

In Section 1.4, we turn to the problem of *typechecking*: given a context Γ, a term e, and a type A, we must either construct evidence for the judgment $\Gamma \vdash e : A$ or conclude that this is not possible.

Depending on the type system in question, different typechecking methods may apply. One of the most widely adopted approaches is *bidirectional typechecking*. A bidirectional typechecker divides the process into two complementary modes: *synthesis*, where the type

of an expression is inferred from its form and context, and *checking*, where we verify that an expression has a given type. To formalize this approach, we extend the language with annotated terms of the form $(e : A)$, where e is an term and A is a type. These annotations serve as local hints that facilitate synthesis in cases where inference alone would be insufficient. We using the following two judgments:

$$\Gamma \vdash e \Rightarrow A \quad \textit{“in context } \Gamma\textit{, } e \textit{ synthesizes type } A\textit{”}$$
$$\Gamma \vdash e \Leftarrow A \quad \textit{“in context } \Gamma\textit{, } e \textit{ checks against type } A\textit{”}$$

Each of the typing rules is now formulated in one of these two modes. We also introduce two additional rules: one to switch from checking to synthesis (`T-sub`), and one to extract the type from an annotation (`T-anno`). The bidirectional rules are:

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash (e : A) \Rightarrow A}\ \texttt{T-anno} \qquad \frac{\Gamma \vdash e \Rightarrow A}{\Gamma \vdash e \Leftarrow A}\ \texttt{T-sub}$$

$$\frac{}{\Gamma, x{:}A, \Gamma' \vdash x \Rightarrow A}\ \texttt{T-var}$$

$$\frac{\Gamma, x{:}A \vdash e \Leftarrow B}{\Gamma \vdash \lambda x.\, e \Leftarrow A \to B}\ \texttt{T-lam} \qquad \frac{\Gamma \vdash e_1 \Rightarrow A \to B \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\ e_2 \Rightarrow B}\ \texttt{T-app}$$

These rules enable a straightforward implementation of a typechecker. The section concludes by describing how we have implemented it in Haskell. The complete source code for the full language is available in the repository accompanying this work (see López-Aquino, 2025).

The final section of this chapter, Section 1.5, presents the operational semantics and theorems that guarantee that well-typed programs cannot go wrong. We conclude by outlining the implementation of a corresponding interpreter in Haskell.

At this point, one may ask: *Is it possible to extend the Curry–Howard correspondence to logics more expressive than propositional logic?* Chapter 2 shows how to do so by following the work of Pfenning and Davies (2001), who demonstrated how the correspondence also holds in the case of modal logic—a family of logics extending propositional logic by incorporating the modalities of necessity ($\Box$) and possibility ($\Diamond$), which allow us to reason not only about what is true, but also about what must be true or could be true.

We present the natural deduction rules for necessity and possibility in Section 2.1. For instance, the rules for necessity are as follows:

$$\frac{}{\Delta, u{::}A\ \texttt{valid}, \Delta'; \Gamma \vdash A\ \texttt{true}}\ \texttt{hyp}^*_u$$

$$\frac{\Delta; \cdot \vdash A\ \texttt{true}}{\Delta; \Gamma \vdash \Box A\ \texttt{true}}\ \Box\texttt{I} \qquad \frac{\Delta; \Gamma \vdash \Box A\ \texttt{true} \qquad \Delta, u{::}A\ \texttt{valid}; \Gamma \vdash B\ \texttt{true}}{\Delta; \Gamma \vdash B\ \texttt{true}}\ \Box\texttt{E}^u$$

In this extended setting, our hypothetical judgments now take the form $\Delta; \Gamma \vdash A\ \texttt{true}$ or $\Delta; \Gamma \vdash A\ \texttt{poss}$. This formulation introduces a dual context: the first context, Δ, contains assumptions of the form $A\ \texttt{valid}$, while the second, Γ, contains assumptions of the form $A\ \texttt{true}$. The notion of necessity is captured at the judgmental level by $A\ \texttt{valid}$, which is defined as shorthand for $\cdot \vdash A\ \texttt{true}$—that is, the proposition A is true under any circumstances. Similarly, natural deduction rules for possibility are also presented.

As in the propositional case, we annotate our natural deduction rules with proof terms. For the fragment involving necessity, the annotated rules are as follows:

$$\frac{}{\Delta, u{::}A, \Delta'; \Gamma \vdash u : A}\ \texttt{hyp}^*_u$$

$$\frac{\Delta; \cdot \vdash e : A}{\Delta; \Gamma \vdash \mathbf{box}\, e : \Box A}\ \Box\texttt{I} \qquad \frac{\Delta; \Gamma \vdash e_1 : \Box A \qquad \Delta, u{::}A; \Gamma \vdash e_2 : B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 : B}\ \Box\texttt{E}^u$$

As before, we verify that the modal connectives satisfy the property of harmony. In the case of necessity, in particular, we have the following local reduction and expansion rules:

$$\begin{array}{lll} \Delta; \Gamma \vdash \mathbf{let\ box}\ u = \mathbf{box}\, e'\ \mathbf{in}\ e : A & \Longrightarrow_R & \Delta; \Gamma \vdash [\![e'/u]\!]e : A \\ \Delta; \Gamma \vdash e : \Box A & \Longrightarrow_E & \Delta; \Gamma \vdash \mathbf{let\ box}\ u = e\ \mathbf{in\ box}\, u : \Box A \end{array}$$

In Section 2.2, we reformulate these rules in a bidirectional style. The bidirectional typing rules are as follows:

$$\frac{}{\Delta, u{::}A, \Delta'; \Gamma \vdash u \Rightarrow A}\ \texttt{T-mvar}$$

$$\frac{\Delta; \cdot \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{box}\, e \Leftarrow \Box A}\ \texttt{T-box} \qquad \frac{\Delta; \Gamma \vdash e_1 \Rightarrow \Box A \qquad \Delta, u{::}A; \Gamma \vdash e_2 \Leftarrow B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \Leftarrow B}\ \texttt{T-letbox}$$

Finally, in Section 2.3, we present the operational semantics for the extended language.

Chapter 3 completes the development by presenting the full system of Contextual Modal Type Theory, in which the modalities of necessity and possibility are relativized to explicit contexts. In this setting, a proposition of the form $[\Psi]A$ expresses that A holds whenever the assumptions in the context Ψ also hold. We begin by extending the natural deduction rules for modal logic to accommodate these contextual modalities. This involves adapting

the introduction and elimination rules for necessity and possibility presented in Chapter 2 to make the relevant context explicit in the judgment. For example, the rules for the fragment concerning contextual necessity are the following:

$$\frac{\Delta, u{::}A\ \mathtt{valid}[\Psi], \Delta'; \Gamma \vdash \Psi}{\Delta, u{::}A\ \mathtt{valid}[\Psi], \Delta'; \Gamma \vdash A\ \mathtt{true}}\ \mathtt{hyp}^*_u$$

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

$$\frac{\Delta; \Gamma \vdash B_1\ \mathtt{true} \qquad \cdots \qquad \Delta; \Gamma \vdash B_m\ \mathtt{true}}{\Delta; \Gamma \vdash y_1{:}B_1\ \mathtt{true}, \cdots, y_m{:}B_m\ \mathtt{true}}\ \mathtt{ctx}$$

As in earlier chapters, we annotate these rules with proof terms that encode the structure of derivations and support a computational interpretation of proofs. Annotating the previous rules, we obtain the following:

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma : \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash u\langle\sigma\rangle : A}\ \mathtt{hyp}^*_u$$

$$\frac{\Delta; \Psi \vdash e : A}{\Delta; \Gamma \vdash \mathbf{box}\ \Psi.\ e : [\Psi]A}\ \Box\mathtt{I} \qquad \frac{\Delta; \Gamma \vdash e_1 : [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash e_2 : B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 : B}\ \Box\mathtt{E}^u$$

$$\frac{\Delta; \Gamma \vdash e_1 : B_1 \qquad \cdots \qquad \Delta; \Gamma \vdash e_m : B_m}{\Delta; \Gamma \vdash (y_1 \leftarrow e_1, \ldots, y_m \leftarrow e_m) : (y_1{:}B_1, \ldots, y_m{:}B_m)}\ \mathtt{ctx}$$

In Section 3.2, we reformulate the typing rules using bidirectional judgments. For the previous rules, the result is as follows:

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash u\langle\sigma\rangle \Rightarrow A}\ \mathtt{T\text{-}mvar}$$

$$\frac{\Delta; \Psi \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{box}\ \Psi.\ e \Leftarrow [\Psi]A}\ \mathtt{T\text{-}box} \qquad \frac{\Delta; \Gamma \vdash e_1 \Rightarrow [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash e_2 \Leftarrow B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \Leftarrow B}\ \mathtt{T\text{-}letbox}$$

$$\frac{\Delta; \Gamma \vdash e_1 \Leftarrow B_1 \qquad \cdots \qquad \Delta; \Gamma \vdash e_m \Leftarrow B_m}{\Delta; \Gamma \vdash (y_1 \leftarrow e_1, \ldots, y_m \leftarrow e_m) \Leftarrow (y_1{:}B_1, \ldots, y_m{:}B_m)}\ \mathtt{T\text{-}ctx}$$

Finally, in Section 3.3, we present operational semantics rules for CMTT.

1

Foundations: Propositions, Programs, and Types

This chapter lays the foundational groundwork for the rest of the thesis. We begin with an exploration of judgments and propositions in the style of Martin-Löf, which provides the formal setting for intuitionistic logic. We then present intuitionistic propositional logic using natural deduction, ensuring that each pair of introduction and elimination rules satisfies the conditions of local soundness and local completeness—that is, they stand in harmony.

The third section introduces the Curry–Howard correspondence, a key conceptual bridge between logic and computation. Here, propositions are viewed as types, and proofs as programs—an idea that underpins the rest of the thesis. We present the simply-typed lambda calculus as the computational counterpart to intuitionistic propositional logic, explaining how types serve as logical specifications for programs.

With this foundation in place, we turn to the problem of typechecking. We present the bidirectional approach to typechecking, implement it in Haskell, and demonstrate how it scales as new language features are added.

Finally, we address the execution of well-typed programs. We define a small-step operational semantics for our language and prove the fundamental properties of progress and preservation, establishing that typechecking ensures safe execution. We conclude the chapter with a basic interpreter implementation that mirrors the operational semantics rules.

This chapter serves both as a conceptual introduction to the interplay between logic and programming languages and as a concrete development of a minimal functional language, setting the stage for the modal and contextual extensions to come in subsequent chapters.

1.1 Judgments and Propositions

This section introduces the fundamental philosophical notions that will be used throughout this work: judgments, propositions, proof, truth, and verification.

The distinction between judgments and propositions is central to the work of Martin-Löf. In terms of conceptual priority, judgments precede propositions, so we begin by addressing them. A a *judgment*, when understood as an act, is an act of knowing, understanding, or

comprehending; when understood as an object, it is a piece or object of knowledge (Martin-Löf, 1996). If we in fact know a judgment, we call it an *evident judgment*.

Proving a judgment is the act of understanding it—acquiring it as a piece of knowledge—and a *proof* is the act by which we come to possess this knowledge. In this setting, a proof should not be confused with the formal proofs typically found in mathematical logic textbooks. A proof here is the very act that renders a judgment evident to someone. It is, in this sense, equivalent to an inference.

We now turn to the notion of a proposition. Consider an expression A and the judgment

$$A \text{ is a proposition,}$$

which we will abbreviate as A `prop`. In the intuitionistic tradition, a proposition is understood as expressing the expectation or intention of a verification. To know this judgment is to know what counts as a verification of A. For example, consider the proposition:

> "There was a massive power outage in Spain on April 28, 2025."

To understand this proposition, we must know what would verify it—such as news reports, timestamps of power loss, or operator logs. It is thus said that the meaning of a proposition is given by what counts as a *verification* of it. What qualifies as a verification depends on the content of the expression and the standards of evidence within a given domain. In logic, in particular, we are concerned with propositions built using logical constants, and we define their meaning by specifying rules for their verification, as we will do for propositional logic in the next section.

Once we know what it means to verify A—that is, once we know A `prop`—we can consider the judgment

$$A \text{ is true,}$$

abbreviated as A `true`. To know this is to actually possess a verification of A, also known as a *proof of* A (Martin-Löf, 1987). Notice that knowledge of the judgment A `prop` amounts to knowing *what* to do to verify A, whereas knowledge of A `true` is knowing *how* to do it.

The next concept we consider is that of a *hypothetical judgment*—a judgment made under assumptions. We write it in the form

$$J_1, \ldots, J_n \vdash J,$$

where $J_1, \ldots, J_n$ is a list of judgments known as *antecedents* and J is a judgment known as *consequent*. To prove such a judgment, we assume each J_i to be known, and under those assumptions, derive J. Moreover, if we have independent proofs of the judgments J_i, we can substitute them for their uses as hypotheses in the proof of J, obtaining a proof that no longer depends on them.

The first form of hypothetical judgment we consider is

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

where the assumptions are labeled with distinct variables x_i. This judgment expresses that the A is true whenever each A_i is true. We abbreviate the list of assumptions as Γ and refer to it as a *context*, writing the judgment more compactly as $\Gamma \vdash A$ `true`.

With the notions introduced so far, we are now prepared to explain the meaning of the logical connectives in propositional logic, using the system of natural deduction presented in the next section.

1.2 Intuitionistic Propositional Logic

Let us begin by recalling that propositional formulas are given by the grammar

$$A, B := P \mid \bot \mid \top \mid A \wedge B \mid A \vee B \mid A \to B,$$

where P ranges over a set of propositional letters. Additionally, we define $\neg A := A \to \bot$.

To formalize the meaning of logical connectives, we use *natural deduction*, which was introduced by Gentzen (1935) and further studied by Prawitz (1965). The system is composed of several *inference rules* and is presented in Figure 1.1. Each inference rule has the form

$$\frac{J_1 \qquad \cdots \qquad J_n}{J}$$

where $J_1, \ldots, J_n$ are judgments called *premises* and J is another judgment called the *conclusion*. An inference rule of this form indicates that if each premise is known, then the conclusion can also be asserted.

A few remarks on the natural deduction system are in order. First, observe that there are two kinds of rules: the rule $\mathtt{hyp}_x$, which we refer to as a *structural rule*, and the remaining rules, which are called *logical rules*. Each logical rule corresponds to a single logical connective. This modularity ensures that the definition of a connective is independent of the others. Second, for each connective, the system provides both introduction and elimination rules. In the introduction rules, the connective appears in the conclusion; in the elimination rules, it appears in a premise. We may thus regard the introduction rules as defining the meaning of the connective, and the elimination rules as governing its use.

Using these rules, we can construct *derivation* or *proof trees*—structured proofs where each step is justified by an inference rule applied to earlier steps. The leaves of the tree correspond to hypotheses, internal nodes to rule applications, and the root to the final conclusion being derived.

We now turn to some examples. In the derivations that follow, we abbreviate assumptions of the form $x{:}A$ `true` simply as $x{:}A$. This notation anticipates the next section, where we introduce a new kind of judgment, $e{:}A$, asserting that e is a proof term of the proposition A. As a notational convention, we assume that $\wedge$ binds more tightly than $\vee$, which in turn binds more tightly than $\to$. In addition, $\to$ associates to the right. Hence, we write $A \vee B \wedge C \to D$ for $(A \vee (B \wedge C)) \to D$ and $A \to B \to C$ for $A \to (B \to C)$.

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

$$\frac{}{\Gamma \vdash \top\ \mathtt{true}}\ \top\mathtt{I} \qquad \frac{\Gamma \vdash \bot\ \mathtt{true}}{\Gamma \vdash A\ \mathtt{true}}\ \bot\mathtt{E}$$

$$\frac{\Gamma, x{:}A\ \mathtt{true} \vdash B\ \mathtt{true}}{\Gamma \vdash A \to B\ \mathtt{true}}\ \to\mathtt{I}^x \qquad \frac{\Gamma \vdash A \to B\ \mathtt{true} \qquad \Gamma \vdash A\ \mathtt{true}}{\Gamma \vdash B\ \mathtt{true}}\ \to\mathtt{E}$$

$$\frac{\Gamma \vdash A\ \mathtt{true} \qquad \Gamma \vdash B\ \mathtt{true}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\ \wedge\mathtt{I}$$

$$\frac{\Gamma \vdash A \wedge B\ \mathtt{true}}{\Gamma \vdash A\ \mathtt{true}}\ \wedge\mathtt{E}_\mathtt{L} \qquad \frac{\Gamma \vdash A \wedge B\ \mathtt{true}}{\Gamma \vdash B\ \mathtt{true}}\ \wedge\mathtt{E}_\mathtt{R}$$

$$\frac{\Gamma \vdash A\ \mathtt{true}}{\Gamma \vdash A \vee B\ \mathtt{true}}\ \vee\mathtt{I}_\mathtt{L} \qquad \frac{\Gamma \vdash B\ \mathtt{true}}{\Gamma \vdash A \vee B\ \mathtt{true}}\ \vee\mathtt{I}_\mathtt{R}$$

$$\frac{\Gamma \vdash A \vee B\ \mathtt{true} \qquad \Gamma, x{:}A\ \mathtt{true} \vdash C\ \mathtt{true} \qquad \Gamma, y{:}B\ \mathtt{true} \vdash C\ \mathtt{true}}{\Gamma \vdash C\ \mathtt{true}}\ \vee\mathtt{E}^{x,y}$$

Figure 1.1: Natural deduction for propositional logic

Example 1.2.1.

- $\vdash A \to A\ \mathtt{true}$.

$$\frac{\dfrac{}{x{:}A \vdash A\ \mathtt{true}}\ \mathtt{hyp}_x}{\vdash A \to A\ \mathtt{true}}\ \to\mathtt{I}^x$$

- $\vdash A \to B \to A\ \mathtt{true}$.

$$\frac{\dfrac{\dfrac{}{x{:}A, y{:}B \vdash A\ \mathtt{true}}\ \mathtt{hyp}_x}{x{:}A \vdash B \to A\ \mathtt{true}}\ \to\mathtt{I}^x}{\vdash A \to B \to A\ \mathtt{true}}\ \to\mathtt{I}^x$$

- $\vdash A \wedge B \to B \wedge A\ \mathtt{true}$.

$$\frac{\dfrac{\dfrac{\dfrac{}{x{:}A \wedge B \vdash A \wedge B\ \mathtt{true}}\ \mathtt{hyp}_x}{x{:}A \wedge B \vdash B\ \mathtt{true}}\ \wedge\mathtt{E}_\mathtt{R} \qquad \dfrac{\dfrac{}{x{:}A \wedge B \vdash A \wedge B\ \mathtt{true}}\ \mathtt{hyp}_x}{x{:}A \wedge B \vdash A\ \mathtt{true}}\ \wedge\mathtt{E}_\mathtt{L}}{x{:}A \wedge B \vdash B \wedge A\ \mathtt{true}}\ \wedge\mathtt{I}}{\vdash A \wedge B \to B \wedge A\ \mathtt{true}}\ \mathtt{hyp}_x$$

- $\vdash A \to \neg\neg A$ true.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}A, y{:}A \to \bot \vdash A \to \bot \text{ true}}\,\mathtt{hyp}_y \quad \dfrac{}{x{:}A, y{:}A \to \bot \vdash A \text{ true}}\,\mathtt{hyp}_x}{x{:}A, y{:}A \to \bot \vdash \bot \text{ true}}\,{\to}\mathtt{E}}{x{:}A \vdash (A \to \bot) \to \bot \text{ true}}\,{\to}\mathtt{I}^y}{\vdash A \to (A \to \bot) \to \bot \text{ true}}\,{\to}\mathtt{I}^x$$

- $\vdash (A \to B \to C) \to (A \wedge B \to C)$ true.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}A \to B \to C, y{:}A \wedge B \vdash A \to B \to C \text{ true}}\,\mathtt{hyp}_x \quad \dfrac{\dfrac{}{x{:}A \to B \to C, y{:}A \wedge B \vdash A \wedge B \text{ true}}\,\mathtt{hyp}_y}{x{:}A \to B \to C, y{:}A \wedge B \vdash A \text{ true}}\,{\wedge}\mathtt{E}_\mathtt{L}}{x{:}A \to B \to C, y{:}A \wedge B \vdash B \to C \text{ true}}\,{\to}\mathtt{E} \quad \dfrac{\dfrac{}{x{:}A \to B \to C, y{:}A \wedge B \vdash A \wedge B \text{ true}}\,\mathtt{hyp}_y}{x{:}A \to B \to C, y{:}A \wedge B \vdash B \text{ true}}\,{\wedge}\mathtt{E}_\mathtt{R}}{x{:}A \to B \to C, y{:}A \wedge B \vdash C \text{ true}}\,{\to}\mathtt{E}}{x{:}A \to B \to C \vdash A \wedge B \to C \text{ true}}\,{\to}\mathtt{I}^y}{\vdash (A \to B \to C) \to (A \wedge B \to C) \text{ true}}\,{\to}\mathtt{I}^x$$

Let us now contrast the system we have introduced with a more traditional approach: *Hilbert-style systems*. Such systems rely on a set of axiom schemata together with a small number of inference rules—often just *modus ponens*. As an example, consider the following axiom schemata:

$$\begin{aligned}
\mathtt{A}_1 &: \quad A \to B \to A \\
\mathtt{A}_2 &: \quad (A \to B \to C) \to (A \to B) \to (A \to C) \\
\mathtt{A}_3 &: \quad (A \to B) \to (\neg B \to \neg A)
\end{aligned}$$

together with the rule of modus ponens:

$$\frac{A \to B \quad A}{B},$$

which corresponds to our implication elimination rule introduced before, $\to\mathtt{E}$. A proof of a formula B in this system consists of a sequence of formulas $A_1, \ldots, A_n, B$ where each formula is either an instance of an axiom schema or follows from earlier formulas by modus ponens. For example, the following sequence is a proof of $A \to A$. For comparison, consider the natural deduction proof for this proposition given in Example 1.2.1.

1. $(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$	instance of $\mathtt{A}_2$
2. $A \to ((A \to A) \to A)$	instance of $\mathtt{A}_1$
3. $(A \to (A \to A)) \to (A \to A)$	1, 2 by modus ponens
4. $A \to (A \to A)$	instance of $\mathtt{A}_1$
5. $A \to A$	3, 4 by modus ponens

Writing Hilbert-style proofs tends to be less intuitive than constructing natural deduction proof trees, as it is common to find oneself midway through a proof uncertain about which axiom schema or inference rule to apply. In contrast, natural deduction offers more guidance:

the choice of rule is usually determined by the main connective of the formula being proved or by applying an elimination rule to formulas in the hypotheses.

Before moving on, it is worth noting that natural deduction proofs can be presented using various notations. For example, *Fitch-style proofs* arrange derivations in a linear, indented format that explicitly tracks assumptions and subproofs. Although differing in presentation, this alternative style preserves the essential structure of natural deduction and hence generally produces yields that are clearer and more transparent than those in Hilbert-style systems.

The logic defined by the rules in Figure 1.1 is *intuitionistic*. It differs from *classical logic* in several important respects—most notably, in that it does not validate certain principles such as the law of excluded middle, expressed by the formula $A \vee \neg A$. In intuitionistic logic, this formula is not derivable. This reflects a deep difference in the interpretation of truth: in classical logic, every proposition is either true or false, independently of whether we can establish which. In intuitionistic logic, by contrast, a proposition is considered true only when we possess a proof of it. Asserting $A \vee \neg A$ would thus require, for any proposition A, a proof either of A or of $\neg A$—yet there are several mathematical statements for which we currently have neither.

Returning to the natural deduction rules, one may reasonably ask: how do we know that the rules for our logical connectives actually make sense? What stops us from inventing arbitrary connectives with ill-defined or inconsistent inference rules? To illustrate this concern, consider the infamous example proposed by Prior (1960), who introduced a fictional connective he called *tonk*, denoted here by 🐟. This connective is defined by an introduction rule that mimics disjunction introduction on the left:

$$\frac{\Gamma \vdash A\ \mathtt{true}}{\Gamma \vdash A\ \text{🐟}\ B\ \mathtt{true}}\ \text{🐟}\mathtt{I}$$

and an elimination rule that behaves like conjunction elimination on the right:

$$\frac{\Gamma \vdash A\ \text{🐟}\ B\ \mathtt{true}}{\Gamma \vdash B\ \mathtt{true}}\ \text{🐟}\mathtt{E}$$

These rules allow us to construct the following derivation:

$$\dfrac{\dfrac{\dfrac{}{x{:}A\ \mathtt{true} \vdash A\ \mathtt{true}}\,\mathtt{hyp}_x}{x{:}A\ \mathtt{true} \vdash A\ \text{🐟}\ B\ \mathtt{true}}\ \text{🐟}\mathtt{I}}{x{:}A\ \mathtt{true} \vdash B\ \mathtt{true}}\ \text{🐟}\mathtt{E}}{\vdash A \to B\ \mathtt{true}}\ {\to}\mathtt{I}^x$$

This derivation shows that from arbitrary A we may infer any B whatsoever. Of greater concern is the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\vdash \top\ \mathtt{true}}\,\top\mathtt{I}}{\vdash \top\ \text{🐟}\ \bot\ \mathtt{true}}\ \text{🐟}\mathtt{I}}{\vdash \bot\ \mathtt{true}}\ \text{🐟}\mathtt{E}}{\vdash A\ \mathtt{true}}\ \bot\mathtt{E}$$

which shows that introducing tonk causes the whole system to collapse into triviality, as anything can be derived within it. In response to Prior's example, Belnap (1962) observed that introducing such a connective would be *"like adding to cricket a player whose role was so specified as to make it impossible to distinguish winning from losing."* Belnap further argued that the idea that inference rules determine meaning is not to be dismissed on the basis of Prior's example because not every set of introduction and elimination rules can be accepted uncritically—there must be constraints.

This leads us to the notion of *harmony* (Dummett, 1991)—the idea that the introduction and elimination rules for a logical connective must be in balance. There are various interpretations of what this balance should be (Francez and Dyckhoff, 2012). In this work, we adopt the approach formulated by Davies and Pfenning (2001), which characterizes harmony in terms of two key properties: *local soundness* and *local completeness*. These notions are also explored in more detail in the lecture notes by Pfenning (2009).

Local soundness ensures that elimination rules do not go beyond what is justified by their corresponding introduction rules. This is demonstrated by showing that when an elimination rule is applied immediately after an introduction rule—that is, the derivation contains a *detour*—we can obtain a derivation of the same conclusion without the detour. To discuss this, we use the notation

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\ \mathtt{true}\end{array}$$

to denote a derivation tree concluding with the judgment $\Gamma \vdash A\ \mathtt{true}$ and refer to the whole derivation tree by $\mathcal{D}$. We use the notation

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\ \mathtt{true}\end{array} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{D}'\\ \Gamma \vdash A\ \mathtt{true}\end{array}$$

for a *local reduction* of derivation $\mathcal{D}$ to derivation $\mathcal{D}'$. Local soundness is then witnessed by a reduction of a derivation containing an introduction rule followed by an elimination to another derivation without the detour. Let us consider this for conjunction. As we have two introduction rules, we exhibit two local reductions:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\ \mathtt{true}\end{array} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma \vdash B\ \mathtt{true}\end{array}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\,\wedge\mathtt{I}}{\Gamma \vdash A\ \mathtt{true}}\,\wedge\mathtt{E_L} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\ \mathtt{true}\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\ \mathtt{true}\end{array} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma \vdash B\ \mathtt{true}\end{array}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\,\wedge\mathtt{I}}{\Gamma \vdash B\ \mathtt{true}}\,\wedge\mathtt{E_R} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{E}\\ \Gamma \vdash B\ \mathtt{true}\end{array}$$

Coming back to the putative connective tonk, we see that local soundness fails, as there is no way to reduce the following derivation:

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A\ \mathtt{true}}}{\Gamma \vdash A \mathbin{\color{cyan}\text{🐟}} B\ \mathtt{true}}\ \color{cyan}\text{🐟}\,\mathtt{I}\qquad \Gamma \vdash B\ \mathtt{true}\ \color{cyan}\text{🐟}\,\mathtt{E}$$

Local completeness, on the other hand, ensures that the elimination rules are not too weak with respect to the introduction rules. To demonstrate this, we show that there is a way to apply the elimination rules so that we can reconstruct a proof of the original conclusion. This is captured by a *local expansion* of the form

$$\dfrac{\mathcal{D}}{\Gamma \vdash A\ \mathtt{true}} \Longrightarrow_E \dfrac{\mathcal{D}'}{\Gamma \vdash A\ \mathtt{true}}$$

where the conclusion contains the logical connective in question, and the derivation $\mathcal{D}'$ has eliminated and then introduced the connective. The following local expansion shows local completeness for conjunction:

$$\dfrac{\mathcal{D}}{\Gamma \vdash A \wedge B\ \mathtt{true}} \Longrightarrow_E \dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\wedge\mathtt{E_L}}{\Gamma \vdash A\ \mathtt{true}} \qquad \dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\wedge\mathtt{E_R}}{\Gamma \vdash B\ \mathtt{true}}}{\Gamma \vdash A \wedge B\ \mathtt{true}}\wedge\mathtt{I}$$

The remainder of this section will be devoted to showing local soundness and completeness for the rest of the connectives. For implication, local soundness is witnessed by the reduction

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma, x{:}A \vdash B\ \mathtt{true}}\to\mathtt{I}^x}{\Gamma \vdash A \to B\ \mathtt{true}} \qquad \dfrac{\mathcal{E}}{\Gamma \vdash A\ \mathtt{true}}}{\Gamma \vdash B\ \mathtt{true}}\to\mathtt{E} \Longrightarrow_R \dfrac{\mathcal{D}'}{\Gamma \vdash B\ \mathtt{true}}$$

where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by substituting the uses of the hypothesis $A\ \mathtt{true}$ by $\mathcal{E}$. Local completeness, on the other hand, is guaranteed by the expansion

$$\dfrac{\mathcal{D}}{\Gamma \vdash A \to B\ \mathtt{true}} \Longrightarrow_E \dfrac{\dfrac{\mathcal{D}'}{\Gamma, x{:}A \vdash A \to B\ \mathtt{true}} \qquad \dfrac{}{\Gamma, x{:}A \vdash A\ \mathtt{true}}\,\mathtt{hyp}_x}{\dfrac{\Gamma, x{:}A \vdash B\ \mathtt{true}}{\Gamma \vdash A \to B\ \mathtt{true}}\to\mathtt{I}^x}\to\mathtt{E}$$

where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by adding the hypothesis $A\ \mathtt{true}$ to every judgment. For disjunction, local soundness is witnessed by the following two reductions:

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash A\ \mathtt{true}}\vee\mathtt{I_L}}{\Gamma \vdash A \vee B\ \mathtt{true}} \qquad \dfrac{\mathcal{E}}{\Gamma, x{:}A \vdash C\ \mathtt{true}} \qquad \dfrac{\mathcal{F}}{\Gamma, y{:}B \vdash C\ \mathtt{true}}}{\Gamma \vdash C\ \mathtt{true}}\vee\mathtt{E}^{x,y} \Longrightarrow_R \dfrac{\mathcal{E}'}{\Gamma \vdash C\ \mathtt{true}}$$

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash B\ \mathtt{true}\end{array}}{\Gamma \vdash A \vee B\ \mathtt{true}}\vee\mathtt{I}_{\mathtt{R}} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma, x{:}A \vdash C\ \mathtt{true}\end{array} \qquad \begin{array}{c}\mathcal{F}\\ \Gamma, y{:}B \vdash C\ \mathtt{true}\end{array}}{\Gamma \vdash C\ \mathtt{true}}\vee\mathtt{E}^{x,y} \Longrightarrow_R \qquad \begin{array}{c}\mathcal{F}'\\ \Gamma \vdash C\ \mathtt{true}\end{array}$$

Local completeness is witnessed by the following expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \vee B\ \mathtt{true}\end{array} \Longrightarrow_E \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \vee B\ \mathtt{true}\end{array} \qquad \dfrac{\dfrac{}{\Gamma, x{:}A \vdash A\ \mathtt{true}}\mathtt{hyp}_x}{\Gamma, x{:}A \vdash A \vee B\ \mathtt{true}}\vee\mathtt{I}_{\mathtt{L}} \qquad \dfrac{\dfrac{}{\Gamma, y{:}B \vdash B\ \mathtt{true}}\mathtt{hyp}_y}{\Gamma, y{:}B \vdash A \vee B\ \mathtt{true}}\vee\mathtt{I}_{\mathtt{R}}}{\Gamma \vdash A \vee B\ \mathtt{true}}\vee\mathtt{E}^{x,y}$$

Finally, observe that although the constants $\top$ and $\bot$ have only an introduction and elimination rule, respectively, we can still carry out the following expansions:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \top\ \mathtt{true}\end{array} \Longrightarrow_E \dfrac{}{\Gamma \vdash \top\ \mathtt{true}}\top\mathtt{I}$$

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \bot\ \mathtt{true}\end{array} \Longrightarrow_E \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \bot\ \mathtt{true}\end{array}}{\Gamma \vdash \bot\ \mathtt{true}}\bot\mathtt{E}$$

Having clarified what it means for inference rules to be well-behaved—that is, in balance through local soundness and completeness—we now turn to their computational interpretation. In the next section, we enrich our natural deduction system with proof terms, paving the way to the Curry–Howard correspondence between logic and computation.

1.3 The Curry-Howard Correspondence

In the previous section, we illustrated how to construct derivation trees in natural deduction. While these trees make the structure of a proof fully explicit, they can be unwieldy to construct and inspect, especially for larger proofs. A more concise alternative is to annotate derivations with *proof terms*. These terms mirror the structure of the derivation and—crucially—the term at the conclusion of an annotated tree contains enough information to reconstruct the entire tree. We begin by introducing the first class of proof terms we will work with, namely *expressions*, defined by the following grammar:

$$e := x \mid () \mid \mathbf{abort}\ e \mid \lambda x.\, e \mid e_1\ e_2 \mid \langle e_1, e_2 \rangle \mid \mathbf{fst}\ e \mid \mathbf{snd}\ e \mid \mathbf{inl}\ e \mid \mathbf{inr}\ e \mid \mathbf{case}(e, x_1.e_1, x_2.e_2)$$

Each of the previous expressions is introduced to correspond one-to-one with an inference rule of our natural deduction system. To formalize this correspondence, we introduce the judgment

$$e : A$$

which states that e is a proof term for the judgment $A\ \mathtt{true}$. The inference rules for this judgment are presented in Figure 1.2, where expressions are highlighted in green and propositions in blue.

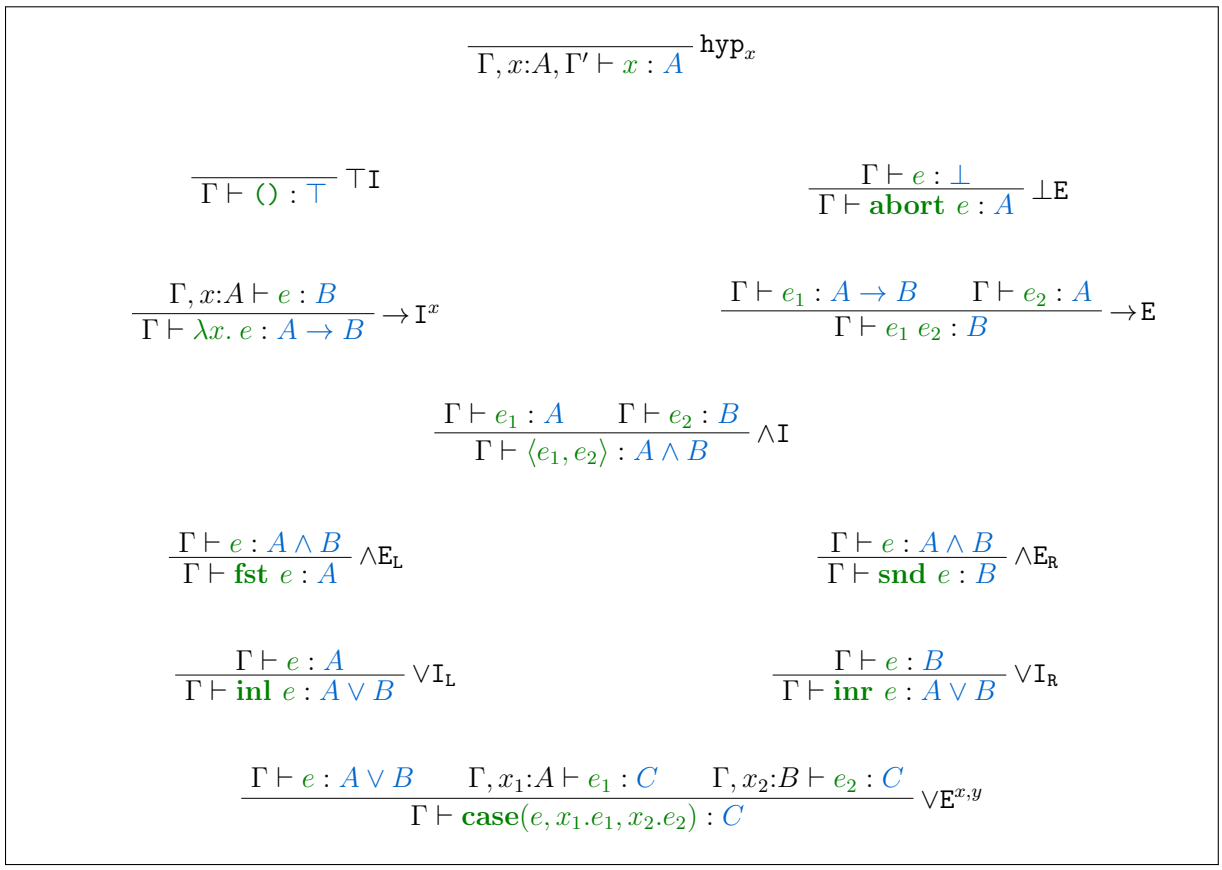

Figure 1.2: Proof term assignment for natural deduction rules

Let us revisit the derivation tree for the judgment $\vdash A \to B \to A$ `true` the we saw in the previous section:

$$\dfrac{\dfrac{\dfrac{}{x{:}A, y{:}B \vdash A\ \mathtt{true}}\,\mathtt{hyp}_x}{x{:}A \vdash B \to A\ \mathtt{true}}\,\to\mathtt{I}^y}{\vdash A \to B \to A\ \mathtt{true}}\,\to\mathtt{I}^x$$

Annotating it with proof terms yields:

$$\dfrac{\dfrac{\dfrac{}{x{:}A, y{:}B \vdash x{:}A}\,\mathtt{hyp}_x}{x{:}A \vdash \lambda y.\, x : B \to A}\,\to\mathtt{I}^y}{\vdash \lambda x.\, \lambda y.\, x : A \to B \to A}\,\to\mathtt{I}^x$$

Notice that the structure of the derivation is now fully captured by the tree's last judgment

$$\vdash \lambda x.\, \lambda y.\, x : A \to B \to A.$$

This compact representation makes the judgment evident from the terms it contains, without requiring a reconstruction of the entire derivation tree. Such a judgment is said to be

analytic. By contrast, the judgment $\vdash A \to B \to A$ `true` is *synthetic*, since verifying it requires constructing or referring to evidence beyond what the judgment's terms alone reveal (Martin-Löf, 1994).

The annotations we have seen so far not only streamline natural deduction proofs but also lie at the core of the *Curry–Howard correspondence*. Under this correspondence, propositions are regarded as *types*, and proof terms as *programs*. Thus, a judgment such as

$$\vdash \lambda x.\, \lambda y.\, x : A \to B \to A$$

is not only a judgment exhibiting that proposition A is true—it is also a well-typed program in a functional programming language. In fact, the system we've presented corresponds to the *simply-typed lambda calculus*, extended with constructs for truth, falsity, conjunction, and disjunction. Each logical connective has a direct type-theoretic counterpart, and each inference rule carries a computational interpretation. For example, our modus ponens rule, $\to$`E`, specifies when function application is permitted.

Before delving deeper into this correspondence, we need to introduce some notions concerning terms. We begin with the notion of a *free variable*, defined inductively as follows:

$$\begin{aligned}
&\mathtt{FV}(x) && = \{x\} \\
&\mathtt{FV}(\mathtt{()}) && = \emptyset \\
&\mathtt{FV}(\textbf{abort}\ e) && = \mathtt{FV}(e) \\
&\mathtt{FV}(\lambda x.\, e) && = \mathtt{FV}(e) \setminus \{x\} \\
&\mathtt{FV}(e_1\ e_2) && = \mathtt{FV}(e_1) \cup \mathtt{FV}(e_2) \\
&\mathtt{FV}(\langle e_1, e_2 \rangle) && = \mathtt{FV}(e_1) \cup \mathtt{FV}(e_2) \\
&\mathtt{FV}(\textbf{fst}\ e) && = \mathtt{FV}(e) \\
&\mathtt{FV}(\textbf{snd}\ e) && = \mathtt{FV}(e) \\
&\mathtt{FV}(\textbf{inl}\ e) && = \mathtt{FV}(e) \\
&\mathtt{FV}(\textbf{inr}\ e) && = \mathtt{FV}(e) \\
&\mathtt{FV}(\textbf{case}(e, x_1.e_1, x_2.e_2)) && = \mathtt{FV}(e) \cup (\mathtt{FV}(e_1) \setminus \{x_1\}) \cup (\mathtt{FV}(e_2) \setminus \{x_2\})
\end{aligned}$$

We can replace the free occurrences of a variable x in an expression e' with another expression e. This operation is called *substitution*, and is denoted by $[e/x]e'$. We define substitutions inductively as follows:

$$
\begin{array}{ll}
[e/x]x & = e \\
[e/x]y & = y \quad \text{if } y \neq x \\
[e/x]() & = () \\
[e/x]\textbf{abort}\ e' & = \textbf{abort}\ [e/x]e' \\
[e/x]\lambda y.\, e' & = \lambda y.\, [e/x]e' \quad \text{if } y \notin \mathtt{FV}(e) \text{ and } y \neq x \\
[e/x](e_1\ e_2) & = ([e/x]e_1)\ ([e/x]e_2) \\
[e/x]\langle e_1, e_2\rangle & = \langle [e/x]e_1, [e/x]e_2\rangle \\
[e/x]\textbf{fst}\ e' & = \textbf{fst}\ [e/x]e' \\
[e/x]\textbf{snd}\ e' & = \textbf{snd}\ [e/x]e' \\
[e/x]\textbf{inl}\ e' & = \textbf{inl}\ [e/x]e' \\
[e/x]\textbf{inr}\ e' & = \textbf{inr}\ [e/x]e' \\
[e/x]\textbf{case}(e', y_1.e_1, y_2.e_2) & = \textbf{case}([e/x]e', y_1.[e/x]e_1, y_2.[e/x]e_2)
\end{array}
$$

In the last line, the side conditions are that x is neither y_1 nor y_2, and that $y_1 \notin \mathtt{FV}(e)$ and $y_2 \notin \mathtt{FV}(e)$. These conditions are analogous to those in the case of lambda abstractions, i.e. expressions of the form $\lambda x.\, e$, and are imposed to *avoid variable capture*. To illustrate why this is necessary, consider the following example. Suppose we want to compute the substitution $[y/x](\lambda y.x)$. If we naively replace x with y, we obtain:

$$[y/x](\lambda y.x) = \lambda y.y.$$

But this is incorrect—the resulting expression has a different meaning: the original term is a constant function returning the free variable x, while the result is the identity function. To avoid this, we first *rename* the bound variable y in $\lambda y.\, x$ to a fresh variable z obtaining the term $\lambda z.x$. Then we apply the substitution safely: $[y/x](\lambda z.x) = \lambda z.y$. This preserves the intended semantics and avoids accidental binding of the substituted variable.

The following theorem ensures that substitution behaves well with respect to typing and will be used later to prove other results.

Theorem 1.3.1 (Substitution Lemma for Truth)**.** If $\Gamma \vdash e' : A$ and $\Gamma, x{:}A \vdash e : B$, then $\Gamma \vdash [e'/x]e : B$.

Proof. By rule induction on the derivation of $\Gamma, x{:}A \vdash e : B$.

- **Case** $\mathtt{hyp}_y$**.** We distinguish two subcases:
  - If $e = x$, then $[e'/x]x = e'$ and we must have $A = B$. Since $\Gamma \vdash e' : A$ by assumption, the result follows.

– If $e = y \neq x$, then $[e'/x]y = y$. But then $y : B$ must be in Γ, so we have $\Gamma \vdash y : B$.

- **Case $\to\mathtt{I}^y$.** In this case, we have a derivation of the form

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma, x{:}A, y{:}C \vdash e'' : D\end{array}}{\Gamma, x{:}A \vdash \lambda y.\, e'' : C \to D}$$

  Using the inductive hypothesis on the judgment $\Gamma, x{:}A, y{:}C \vdash e'' : D$ allows us to deduce that $\Gamma, y{:}C \vdash [e'/x]e'' : D$. Now, by the rule $\to\mathtt{I}^y$, we see that $\Gamma \vdash \lambda y.\, [e'/x]e''$, as desired.

- **Case $\to\mathtt{E}$.** Here, our derivation has the form

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma, x{:}A \vdash e_1 : C \to B\end{array} \qquad \begin{array}{c}\vdots\\ \Gamma, x{:}A \vdash e_2 : C\end{array}}{\Gamma, x{:}A \vdash e_1\ e_2 : B}$$

  Using the inductive hypothesis on both premises, we deduce that $\Gamma \vdash [e'/x]e_1 : C \to B$ and $\Gamma \vdash [e'/x]e_2 : C$. Now using the rule $\to\mathtt{E}$, it follows that $\Gamma \vdash ([e'/x]e_1)([e'/x]e_2) : B$, as desired.

All other cases follow similarly by applying the inductive hypothesis to subterms and reconstructing the typing derivation using the corresponding typing rule. □

Let us now return to the correspondence between natural deduction derivations and terms in the lambda calculus. This correspondence is more than a notational convenience: it reflects a deep structural relationship. In particular, the rules of *harmony*—which ensure that the introduction and elimination rules for each connective are properly balanced—admit a computational interpretation as well. We begin to explore this connection by considering the following proof tree for the judgment $\vdash (A \to B) \to (B \to C) \to (A \to C)$ `true`:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f{:}A \to B, g{:}B \to C, x{:}A \vdash B \to C\ \mathtt{true}}\,\mathtt{hyp}_g \quad \dfrac{\dfrac{}{f{:}A \to B, \ldots \vdash A \to B\ \mathtt{true}}\,\mathtt{hyp}_f \quad \dfrac{}{\ldots, x{:}A \vdash A\ \mathtt{true}}\,\mathtt{hyp}_x}{f{:}A \to B, g{:}B \to C, x{:}A \vdash B\ \mathtt{true}}\to\mathtt{E}}{f{:}A \to B, g{:}B \to C, x{:}A \vdash C\ \mathtt{true}}\to\mathtt{E}}{f{:}A \to B, g{:}B \to C \vdash A \to C\ \mathtt{true}}\to\mathtt{I}^x}{f{:}A \to B \vdash (B \to C) \to (A \to C)\ \mathtt{true}}\to\mathtt{I}^g}{\vdash (A \to B) \to (B \to C) \to (A \to C)\ \mathtt{true}}\to\mathtt{I}^f$$

Now we annotate the previous derivation tree with proof terms:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f{:}A \to B, g{:}B \to C, x{:}A \vdash g : B \to C}\,\mathtt{hyp}_g \quad \dfrac{\dfrac{}{f{:}A \to B, \ldots \vdash f : A \to B}\,\mathtt{hyp}_f \quad \dfrac{}{\ldots, x{:}A \vdash x : A}\,\mathtt{hyp}_x}{f{:}A \to B, g{:}B \to C, x{:}A \vdash f\ x : B}\to\mathtt{E}}{f{:}A \to B, g{:}B \to C, x{:}A \vdash g\ f\ x : C}\to\mathtt{E}}{f{:}A \to B, g{:}B \to C \vdash \lambda x.\, g\ f\ x : A \to C}\to\mathtt{I}^x}{f{:}A \to B \vdash \lambda g.\, \lambda x.\, g\ f\ x : (B \to C) \to (A \to C)}\to\mathtt{I}^g}{\vdash \lambda f.\, \lambda g.\, \lambda x.\, g\ f\ x : (A \to B) \to (B \to C) \to (A \to C)}\to\mathtt{I}^f$$

In computational terms, the program $\lambda f.\ \lambda g.\ \lambda x.\ g\ f\ x$ computes the composition of two functions f and g. Now let us consider the case where both f and g are provided and have type $A \to A$. Intuitively, the result should also be a function having type $A \to A$. To see this, notice that we can modify the previous derivation tree by replacing all occurrences of B and C by A. We then add assumptions stating that $\lambda z.\ z$ and $\lambda y.\ y$ both have type $A \to A$, and apply implication elimination twice to construct a derivation tree for the following judgment:

$$\vdash ((\lambda f.\ \lambda g.\ \lambda x.\ g\ f\ x)(\lambda z.\ z))(\lambda y.\ y) : A \to A.$$

This derivation tree, however, contains some *detours*: instances where an implication is first introduced and then immediately eliminated. Since our proof term retains all the information of the derivation, these detours are also visible in it. The first such detour is highlighted in green below:

$$\vdash ((\lambda f.\ \lambda g.\ \lambda x.\ g\ f\ x)(\lambda z.\ z))(\lambda y.\ y) : A \to A.$$

Such an expression is called a *redex* (short for reducible expression) and reflects precisely the pattern of a detour: an introduction immediately followed by an elimination. This pattern was previously discussed in the context of local soundness, where we showed that any derivation containing such a detour can be transformed into a simpler derivation without it via a local reduction. For implication, the local reduction can now be rewritten using proof terms and substitution as

$$(\lambda x.\ e)e' \Longrightarrow_R [e'/x]e,$$

which captures the idea that all occurrences of the hypothesis x are replaced by the proof term e'.

Returning to our example, we can perform a sequence of reductions as follows, where the redex is marked at each step:

$$\begin{aligned}
((\lambda f.\ \lambda g.\ \lambda x.\ g\ f\ x)(\lambda z.\ z))(\lambda y.\ y) &\Longrightarrow_R (\lambda g.\ \lambda x.\ g\ (\lambda z.\ z)\ x)(\lambda y.\ y) \\
&\Longrightarrow_R (\lambda g.\ \lambda x.\ g\ x)(\lambda y.\ y) \\
&\Longrightarrow_R \lambda x.\ (\lambda y.\ y)\ x \\
&\Longrightarrow_R \lambda x.\ x.
\end{aligned}$$

Note that these reductions still correspond to transformations on derivation trees. More precisely, the full reduction should be expressed as

$$\vdash ((\lambda f.\ \lambda g.\ \lambda x.\ g\ f\ x)(\lambda z.\ z))(\lambda y.\ y) : A \to A \quad \Longrightarrow_R^* \quad \vdash \lambda x.\ x : A \to A,$$

where the notation $\Longrightarrow_R^*$ indicates multiple reductions.

A similar analysis can be performed for the rest of the logical connectives, obtaining the following list:

$$\begin{aligned}
\Gamma \vdash (\lambda x.\ e)\ e' : B &\Longrightarrow_R \Gamma \vdash [e'/x]e : B \\
\Gamma \vdash \mathbf{fst}\ \langle e_1, e_2\rangle : A &\Longrightarrow_R \Gamma \vdash e_1 : A \\
\Gamma \vdash \mathbf{snd}\ \langle e_1, e_2\rangle : B &\Longrightarrow_R \Gamma \vdash e_2 : B \\
\Gamma \vdash \mathbf{case}(\mathbf{inl}\ e, x_1.e_1, x_2.e_2) : C &\Longrightarrow_R \Gamma \vdash [e/x_1]e_1 : C \\
\Gamma \vdash \mathbf{case}(\mathbf{inr}\ e, x_1.e_1, x_2.e_2) : C &\Longrightarrow_R \Gamma \vdash [e/x_2]e_2 : C
\end{aligned}$$

These reductions correspond precisely to the well-known concept of *β-reduction* in the lambda calculus. We will return to these ideas in greater detail in the final section of this chapter, where we will outline a strategy for executing programs based on the reductions we have discussed.

Similarly, we can consider local expansions from the perspective of proof terms. The list below corresponds to *η-expansions* in the lambda calculus.

$$\begin{array}{lll}
\Gamma \vdash e : A \wedge B & \Longrightarrow_E & \Gamma \vdash \langle \mathbf{fst}\ e, \mathbf{snd}\ e \rangle : A \wedge B \\
\Gamma \vdash e : A \to B & \Longrightarrow_E & \Gamma \vdash \lambda x.\, e\ x : A \to B \\
\Gamma \vdash M : A \vee B & \Longrightarrow_E & \Gamma \vdash \mathbf{case}(e, x_1.\mathbf{inl}\ x_1, x_2.\mathbf{inr}\ x_2) : A \vee B \\
\Gamma \vdash M : \top & \Longrightarrow_E & \Gamma \vdash () : \top \\
\Gamma \vdash M : \bot & \Longrightarrow_E & \Gamma \vdash \mathbf{abort}\ M : \bot
\end{array}$$

We conclude this section with some additional remarks on the Curry-Howard correspondence. Beyond revealing a deep connection between logic and computation, it can be used as a guiding principle for designing programming languages grounded in logic: here, types become more than mere labels—they act as specifications that describe what a program must accomplish. At the same time, types function as abstractions, concealing unnecessary details while ensuring that essential program properties are maintained. This foundation enables powerful reasoning about our programs: well-typed programs inherently carry proofs of their own correctness and reliability.

As we will explore in the next chapter, the correspondence extends to richer logics, enabling even more expressive, logic-based language features. For now, we shift focus to practical programming concerns, starting with typechecking—the process that verifies whether programs conform to their specified types.

1.4 Typechecking Programs

Let us consider again the judgment $\Gamma \vdash e : A$. Depending on what information we have available, different problems may arise. If, in addition to the context Γ, both the expression e and the type A are given, then we may be asked to verify whether e indeed has type A under context Γ. This is the problem of *typechecking*.

The rules presented earlier in Figure 1.2 can be used to guide typechecking, provided that every subexpression in e is annotated with its type. In practice, however, requiring complete annotations would be overly verbose and impractical.

To alleviate this burden while retaining a sound and effective method for typechecking, a widely adopted approach is *bidirectional typechecking*, which we now describe.

1.4.1 Bidirectional Typechecking

The core idea of bidirectional typechecking is to divide the typing process into two complementary modes: *synthesis*, where the type of an expression is inferred from its structure and the context; and *checking*, where we verify that an expression matches a given type.

To formalize this approach, we extend our set of expressions with explicit annotations of the form $(e : A)$, where e is an expression and A is a type. These annotations serve as local hints that help guide synthesis in contexts where inference alone would be insufficient.

We follow the presentation of Dunfield and Krishnaswami (2021), which surveys and systematizes bidirectional typechecking, although the idea itself predates this work and was presented by Pierce and Turner (1997, 1998, 2000). We make use of the following two judgments:

$$\Gamma \vdash e \Rightarrow A \quad \textit{"under context } \Gamma\textit{, } e \textit{ synthesizes type } A\textit{"}$$

$$\Gamma \vdash e \Leftarrow A \quad \textit{"under context } \Gamma\textit{, } e \textit{ checks against type } A\textit{"}$$

These judgments work together to assign types to expressions in a modular and predictable way. Most of the rules in the bidirectional system correspond directly to those presented earlier in Figure 1.2, reorganized according to whether they operate in synthesis or checking mode. There are two exceptions. The first is the rule `T-anno` for annotated expressions: when we write $(e : A)$, we are asserting that e should have type A, so we check that e indeed has type A, and if it does, we allow the whole expression to synthesize the type A. The second is the subsumption rule `T-sub`, which serves as a bridge between the two modes: when asked to check that e has type B, we first synthesize a type A for e and then compare it with B. This rule allows checking to fall back on synthesis when possible, provided the synthesized type matches the expected one.

The decision to use synthesis or checking is guided by the information available in the typing judgment. Synthesis is used when the context Γ provides sufficient information to determine the type — for example, when typing a variable or applying a function. Checking is used when no such information is available from the context, and we instead validate the expression against a known type.

The set of bidirectional typing rules is given in Figure 1.3.

To illustrate how these rules behave in practice, we now turn to a few examples. These demonstrate how bidirectional typechecking succeeds in typical cases, fails when type information is insufficient, and how it can be guided by annotations when needed.

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash (e : A) \Rightarrow A}\ \texttt{T-anno}$$

$$\frac{\Gamma \vdash e \Rightarrow A}{\Gamma \vdash e \Leftarrow A}\ \texttt{T-sub}$$

$$\frac{}{\Gamma, x{:}A, \Gamma' \vdash x \Rightarrow A}\ \texttt{T-var}$$

$$\frac{}{\Gamma \vdash () \Leftarrow \top}\ \texttt{T-unit} \qquad \frac{\Gamma \vdash e \Leftarrow \bot}{\Gamma \vdash \mathbf{abort}\ e \Leftarrow A}\ \texttt{T-abort}$$

$$\frac{\Gamma, x{:}A \vdash e \Leftarrow B}{\Gamma \vdash \lambda x.\, e \Leftarrow A \to B}\ \texttt{T-lam} \qquad \frac{\Gamma \vdash e_1 \Rightarrow A \to B \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\ e_2 \Rightarrow B}\ \texttt{T-app}$$

$$\frac{\Gamma \vdash e_1 \Leftarrow A \qquad \Gamma \vdash e_2 \Leftarrow B}{\Gamma \vdash \langle e_1, e_2 \rangle \Leftarrow A \wedge B}\ \texttt{T-pair}$$

$$\frac{\Gamma \vdash e \Rightarrow A \wedge B}{\Gamma \vdash \mathbf{fst}\ e \Rightarrow A}\ \texttt{T-fst} \qquad \frac{\Gamma \vdash e \Rightarrow A \wedge B}{\Gamma \vdash \mathbf{snd}\ e \Rightarrow B}\ \texttt{T-snd}$$

$$\frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash \mathbf{inl}\ e \Leftarrow A \vee B}\ \texttt{T-inl} \qquad \frac{\Gamma \vdash e \Leftarrow B}{\Gamma \vdash \mathbf{inr}\ e \Leftarrow A \vee B}\ \texttt{T-inr}$$

$$\frac{\Gamma \vdash e \Rightarrow A \vee B \qquad \Gamma, x_1{:}A \vdash e_1 \Leftarrow C \qquad \Gamma, x_2{:}B \vdash e_2 \Leftarrow C}{\Gamma \vdash \mathbf{case}(e, x_1.e_1, x_2.e_2) \Leftarrow C}\ \texttt{T-case}$$

Figure 1.3: Bidirectional typechecking judgments

Example 1.4.1.

- Below we construct a derivation tree for the judgment $\cdot \vdash \lambda x.\, \mathbf{fst}\ x \Leftarrow A \wedge B \to A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}A \wedge B \vdash x \Rightarrow A \wedge B}\ \texttt{T-var}}{x{:}A \wedge B \vdash \mathbf{fst}\ x \Rightarrow A}\ \texttt{T-fst}}{x{:}A \wedge B \vdash \mathbf{fst}\ x \Leftarrow A}\ \texttt{T-sub}}{\cdot \vdash \lambda x.\, \mathbf{fst}\ x \Leftarrow A \wedge B \to A}\ \texttt{T-lam}$$

- Synthesizing a type for $\lambda x.\, x$ fails: lambda abstractions cannot synthesize a type on their own, since there is no rule for $\lambda x.\, e$ in synthesis mode.

- To resolve the issue in the previous case, we can provide an annotation, and construct a derivation tree for the judgment $\cdot \vdash (\lambda x.\, x : A \to A) \Rightarrow A \to A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}A \vdash x \Rightarrow A}\,\texttt{T-var}}{x{:}A \vdash x \Leftarrow A}\,\texttt{T-sub}}{\cdot \vdash \lambda x.\, x \Leftarrow A \to A}\,\texttt{T-lam}}{\cdot \vdash (\lambda x.\, x : A \to A) \Rightarrow A \to A}\,\texttt{T-anno}$$

We now relate the bidirectional judgments $\Gamma \vdash e \Rightarrow A$ and $\Gamma \vdash e \Leftarrow A$ to the original typing judgment $\Gamma \vdash e : A$ by means of the following two theorems. To state them precisely, we write $|e|$ for the expression e stripped of any type annotations it may contain.

Theorem 1.4.2 (Soundness)**.** If $\Gamma \vdash e \Rightarrow A$ or $\Gamma \vdash e \Leftarrow A$, then $\Gamma \vdash |e| : A$.

Proof. We proceed by mutual rule induction on the derivations of the judgments $\Gamma \vdash e \Rightarrow A$ and $\Gamma \vdash e \Leftarrow A$. To illustrate, we present three cases.

- **Case `T-var`.** Here, we have the following derivation:

$$\dfrac{}{\Gamma, x{:}A, \Gamma' \vdash x \Rightarrow A}$$

  The desired result clearly follows.

- **Case `T-anno`.** In this case, we have a derivation of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash e \Leftarrow A\end{array}}{\Gamma \vdash (e : A) \Rightarrow A}$$

  Applying the inductive hypothesis to the premise yields $\Gamma \vdash |e| : A$, as desired.

- **Case `T-sub`.** Here, the corresponding derivation has the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash e \Rightarrow A\end{array}}{\Gamma \vdash e \Leftarrow A}$$

  Application of the inductive hypothesis immediately yields the desired result.

□

Theorem 1.4.3 (Completeness)**.** If $\Gamma \vdash e : A$, then there exists some e' such that $\Gamma \vdash e' \Leftarrow A$ and $e = |e'|$.

Proof. By rule induction on the derivation of $\Gamma \vdash e : A$. To illustrate, we show the case for the rule $\to$E. Here, we must have a derivation of the following form:

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash e_1 : A \to B & \Gamma \vdash e_2 : A\end{array}}{\Gamma \vdash e_1\, e_2 : B}$$

By the inductive hypothesis, there exists an expression e_1' such that $|e_1'| = e_1$ and $\Gamma \vdash e_1' \Leftarrow A \to B$, so there exists a derivation of the form

$$\begin{array}{c}\mathcal{D}_1 \\ \Gamma \vdash e_1' \Leftarrow A \to B\end{array}$$

Similarly, there is a term e_2' such that $|e_2'| = e_2$, and a derivation of the form

$$\begin{array}{c}\mathcal{D}_2 \\ \Gamma \vdash e_2' \Leftarrow B\end{array}$$

Let $e' = (e_1' : A \to B)e_2$. The following derivation shows that the desired conclusion holds in this case:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma \vdash e_1' \Leftarrow A \to B\end{array}}{\Gamma \vdash (e_1' : A \to B) \Rightarrow A \to B}\,\texttt{T-anno} \qquad \dfrac{\begin{array}{c}\mathcal{D}_2 \\ \Gamma \vdash e_2' \Leftarrow B\end{array}}{}}{\Gamma \vdash (e_1' : A \to B)e_2 \Rightarrow B}\,\texttt{T-app}}{\Gamma \vdash (e_1' : A \to B)e_2 \Leftarrow B}\,\texttt{T-sub}$$

□

To bring our language closer to a real functional programming language, we now extend it with additional constructs. Specifically, we introduce two base types—integers and booleans—along with familiar operations such as arithmetic, comparison, conditionals, let-bindings, and recursion.

The typing rules for these new constructs are given in Figure 1.4, and the corresponding bidirectional typing rules are provided in Figure 1.5. In these figures, $\dot{*}$ and $\dot{\circ}$ range over syntactic operators: $\dot{*} \in \{\dot{+}, \dot{-}, \dot{\times}\}$ for arithmetic, and $\dot{\circ} \in \{\dot{=}, \dot{<}, \dot{>}\}$ for comparison. These symbols with a dot above denote the operations as they appear in the language's syntax. In the next section, we will use their corresponding undotted counterparts to represent the actual semantic operations used during evaluation.

Free variables and substitutions for the new expressions are defined as follows:

$$
\begin{aligned}
&\mathtt{FV}(\overline{n}) && = \varnothing \\
&\mathtt{FV}(\mathtt{true}) && = \varnothing \\
&\mathtt{FV}(\mathtt{false}) && = \varnothing \\
&\mathtt{FV}(e_1 \mathbin{\dot{*}} e_2) && = \mathtt{FV}(e_1) \cup \mathtt{FV}(e_2) \\
&\mathtt{FV}(e_1 \mathbin{\dot{\circ}} e_2) && = \mathtt{FV}(e_1) \cup \mathtt{FV}(e_2) \\
&\mathtt{FV}(\textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2) && = \mathtt{FV}(b) \cup \mathtt{FV}(e_1) \cup \mathtt{FV}(e_2) \\
&\mathtt{FV}(\textbf{let val } x = e_1 \textbf{ in } e_2) && = \mathtt{FV}(e_1) \cup (\mathtt{FV}(e_2) \setminus \{x\}) \\
&\mathtt{FV}(\textbf{fun } f(x) \Rightarrow e) && = \mathtt{FV}(e) \setminus \{f, x\}
\end{aligned}
$$

$$
\begin{aligned}
&[e/x]\overline{n} && = \overline{n} \\
&[e/x]\mathtt{true} && = \mathtt{true} \\
&[e/x]\mathtt{false} && = \mathtt{false} \\
&[e/x](e_1 \mathbin{\dot{*}} e_2) && = ([e/x]e_1) \mathbin{\dot{*}} ([e/x]e_2) \\
&[e/x](e_1 \mathbin{\dot{\circ}} e_2) && = ([e/x]e_1) \mathbin{\dot{\circ}} ([e/x]e_2) \\
&[e/x](\textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2) && = \textbf{if } [e/x]b \textbf{ then } [e/x]e_1 \textbf{ else } [e/x]e_2 \\
&[e/x](\textbf{let val } y = e_1 \textbf{ in } e_2) && = \textbf{let val } y = [e/x]e_1 \textbf{ in } [e/x]e_2 \quad \text{if } y \notin \mathtt{FV}(e) \text{ and } y \neq x \\
&[e/x](\textbf{fun } f(y) \Rightarrow e') && = \textbf{fun } f(y) \Rightarrow [e/x]e' \quad \text{if } f, y \notin \mathtt{FV}(e) \text{ and } f \neq x, y \neq x
\end{aligned}
$$

While many of these constructs could be encoded using the core calculus—for example, booleans as $\top + \top$, conditionals as a special instance of the case construct, or let as syntactic sugar for function application —we choose to treat them as primitive. This avoids choosing a particular encoding and allows us to define their typing behavior directly, which makes examples and reasoning more transparent, even if it slightly expands the rule sets.

The correspondence between the original typing judgment and the bidirectional formulation, as captured by Theorems 1.4.2 and 1.4.3, continues to hold for the enriched language. The proofs extend by applying the same inductive methods as outlined earlier, with additional cases for the new constructs. Similarly, the Substitution Lemma (Theorem 1.3.1) continues to hold.

$$\frac{}{\Gamma \vdash \overline{n} : \texttt{int}}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \dot{*} e_2 : \texttt{int}} \qquad \frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \dot{\circ} e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : B \qquad \Gamma, x{:}B \vdash e_2 : A}{\Gamma \vdash \textbf{let val } x = e_1 \textbf{ in } e_2 : A} \qquad \frac{\Gamma, f{:}A \to B, x{:}A \vdash e : B}{\Gamma \vdash \textbf{fun } f(x) \Rightarrow e : A \to B}$$

$$\frac{\Gamma \vdash b : \texttt{bool} \qquad \Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash \textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2 : A}$$

Figure 1.4: Typing judgments for additional constructs

$$\frac{}{\Gamma \vdash \overline{n} \Rightarrow \texttt{int}}\ \texttt{T-int}$$

$$\frac{}{\Gamma \vdash \texttt{true} \Rightarrow \texttt{bool}}\ \texttt{T-true} \qquad \frac{}{\Gamma \vdash \texttt{false} \Rightarrow \texttt{bool}}\ \texttt{T-false}$$

$$\frac{\Gamma \vdash e_1 \Leftarrow \texttt{int} \qquad \Gamma \vdash e_2 \Leftarrow \texttt{int}}{\Gamma \vdash e_1 \dot{*} e_2 \Rightarrow \texttt{int}}\ \texttt{T-op} \qquad \frac{\Gamma \vdash e_1 \Leftarrow \texttt{int} \qquad \Gamma \vdash e_2 \Leftarrow \texttt{int}}{\Gamma \vdash e_1 \dot{\circ} e_2 \Rightarrow \texttt{bool}}\ \texttt{T-pred}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow B \qquad \Gamma, x{:}B \vdash e_2 \Leftarrow A}{\Gamma \vdash \textbf{let val } x = e_1 \textbf{ in } e_2 \Leftarrow A}\ \texttt{T-let} \qquad \frac{\Gamma, f{:}A \to B, x{:}A \vdash e \Leftarrow B}{\Gamma \vdash \textbf{fun } f(x) \Rightarrow e \Leftarrow A \to B}\ \texttt{T-fun}$$

$$\frac{\Gamma \vdash b \Leftarrow \texttt{bool} \qquad \Gamma \vdash e_1 \Leftarrow A \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash \textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2 \Leftarrow A}\ \texttt{T-cond}$$

Figure 1.5: Bidirectional typechecking judgments for additional constructs

At this point, writing out typing derivations by hand—especially for the extended language—becomes tedious and error-prone. This motivates the implementation of a typechecker that follows the bidirectional rules introduced so far.

1.4.2 Implementing a Typechecker

To accompany this work, we have implemented a bidirectional typechecker in Haskell. In this section, we outline the structure of that implementation. For clarity and brevity, we restrict attention to a minimal core language: the simply-typed lambda calculus with booleans and integers.

We begin by defining the types and expressions used in the implementation, as shown in Code Snippet 1.1.

```haskell
data Type
= BoolTy  -- Booleans
| IntTy    -- Inntegers
| Arrow Type Type  -- Function types
deriving (Eq, Show)

data Term
= Var String  -- Variables
| Lam String Expr  -- Lambda abstractions
| App Expr Expr    -- Application
| LitBool Bool    -- True, false
| LitInt Int    -- Integer literals
| Ann Expr Type    -- Annotations
deriving (Eq, Show)
```

Code Snippet 1.1: Abstract syntax trees for types and expressions

The typechecker is organized around two functions: `synth`, which attempts to infer the type of an expression, and `check`, which verifies that an expression has a specified type. Their signatures and the definition of typing contexts are given in Code Snippet 1.2.

```haskell
type Ctx = [(String, Type)]

-- Synthesis: try to infer the type of an expression
synth :: Ctx -> Term -> Either TypeError Type

-- Checking: verify that an expression has a given type
check :: Ctx -> Term -> Type -> Either TypeError ()
```

Code Snippet 1.2: Typing context and typechecking function signatures

The implementation of these functions, shown in Code Snippet 1.3, closely follows the bidirectional typing rules presented earlier. Each clause corresponds to a specific typing rule, and this correspondence is explicitly indicated by comments in the code.

```haskell
check :: Ctx -> Term -> Type -> Either TypeError ()
-- T-lam
check ctx (Lam x e) t =
  case t of
    Arrow t1 t2 -> check ((x, t1) : ctx) e t2
    _ -> Left (NotAFunctionType t)
-- T-sub
check modCtx ctx e ty = do
  inferred <- synth modCtx ctx e
  if inferred == ty
    then return ()
    else Left TypeMismatch

synth :: Ctx -> Term -> Either TypeError Type
-- T-true, T-false
synth _  (LitBool _) = Right BoolTy
-- T-int
synth _ (LitInt _) = Right IntTy
-- T-app
synth _ (App e1 e2) = do
  t <- synth ctx e1
  case t of
    Arrow t1 t2 -> check ctx e2 t1 >> return t2
    _ -> Left (NotAFunctionType t)
-- T-anno
synth ctx (Ann e ty) = ty <$ check ctx e ty
```

Code Snippet 1.3: Implementation of the core bidirectional typechecking rules

The version presented here is deliberately minimal. A complete version of the typechecker—including support for the full language developed throughout this work—is available in the accompanying code repository.[1]

1.4.3 Other Approaches

The bidirectional approach we have studied so far is one of several possible approaches. The choice of inference rules and information flow typically depends on the type system at hand. To consider another possibility, recall our bidirectional rule for application:

[1] See López-Aquino, 2025.

$$\frac{\Gamma \vdash e_1 \Rightarrow A \to B \qquad \Gamma \vdash e_2 \Leftarrow A}{\Gamma \vdash e_1\ e_2 \Leftarrow B}\ \texttt{T-app}$$

Here, type information flows from the function, e_1, to the argument, e_2—we synthesize the function type $A \to B$ for e_1 first, and then check e_2 against B. Xie and Oliveira (2018) propose an alternative where the direction of the information flow is reversed: we first analyze the type of the argument, and then proceed to type the function. For this purpose, they introduce a judgment of the form:

$$\Gamma \mid \Sigma \vdash e \Rightarrow A \quad \text{"under typing context } \Gamma\text{, and application context } \Sigma\text{, } e \text{ has type } A\text{"}.$$

Here Σ is a stack of types. While analyzing an application expression, we push the type of an argument into Σ to later check that the function indeed accepts arguments of that type. This is captured by the following rule:

$$\frac{\Gamma \vdash e_2 \Rightarrow A \qquad \Gamma \mid \Sigma, A \vdash e_1 \Rightarrow A \to B}{\Gamma \mid \Sigma \vdash e_1\ e_2 \Rightarrow B}\ \texttt{T-app-2}$$

As an example, suppose that we want to typecheck the program $(\lambda x.\ x)\,42$. Notice that this is not possible with our current set of bidirectional rules—we need to annotate the lambda abstraction and instead typecheck the program $(\lambda x.\ x : \texttt{int} \to \texttt{int})\,42$. Consider the following three rules to work in tandem with `T-app-2`:

$$\frac{}{\Gamma, x{:}A \vdash x \Rightarrow A}\ \texttt{T-var-2} \qquad \frac{}{\Gamma \vdash \overline{n} \Rightarrow \texttt{int}}\ \texttt{T-int-2} \qquad \frac{\Gamma, x{:}A \mid \Sigma \vdash e \Rightarrow B}{\Gamma \mid \Sigma, A \vdash \lambda x.\ e \Rightarrow A \to B}\ \texttt{T-lam-2}$$

We can then typecheck the program $(\lambda x.\ x)\,42$ as follows:

$$\frac{\dfrac{}{\vdash 42 \Rightarrow \texttt{int}}\ \texttt{T-int-2} \qquad \dfrac{\dfrac{}{x : \texttt{int} \vdash x \Rightarrow \texttt{int}}\ \texttt{T-var-2}}{\cdot \mid \texttt{int} \vdash \lambda x.\ x \Rightarrow \texttt{int}}\ \texttt{T-lam-2}}{\vdash (\lambda x.\ x)\,42 \Rightarrow \texttt{int}}\ \texttt{T-app-2}$$

1.5 Running Programs

The Curry–Howard correspondence invites us to view proofs as programs and propositions as types. From this perspective, typechecking corresponds to proof verification. But what does it mean to run a program? We define a proof as *normal* if it contains no detours—that is, its annotated form has no β-redexes. Running a program, then, corresponds to *partially* normalizing a proof: applying local reductions to eliminate some redexes, but not necessarily all. The precise mechanics of this process will become clearer as we develop the operational semantics. In this section, we turn to the dynamic side of our language: how programs compute.

1.5.1 Operational Semantics

Before explaining how to run programs, we must define what is means for a program to be done; this is captured by the notion of a *value*: an expression that represents a completed computation and cannot be further reduced. For this purpose, we introduce the judgment e `value`, where e is a closed expression, i.e., an expression with no free variables. The inference rules for this judgment are presented in Figure 1.6 below.

Note that the rules `V-lam` and `V-fun` do not require the body e to be a value. This reflects the fact that evaluation is only partially normalizing: we do not reduce inside function bodies, since that would require handling free variables.

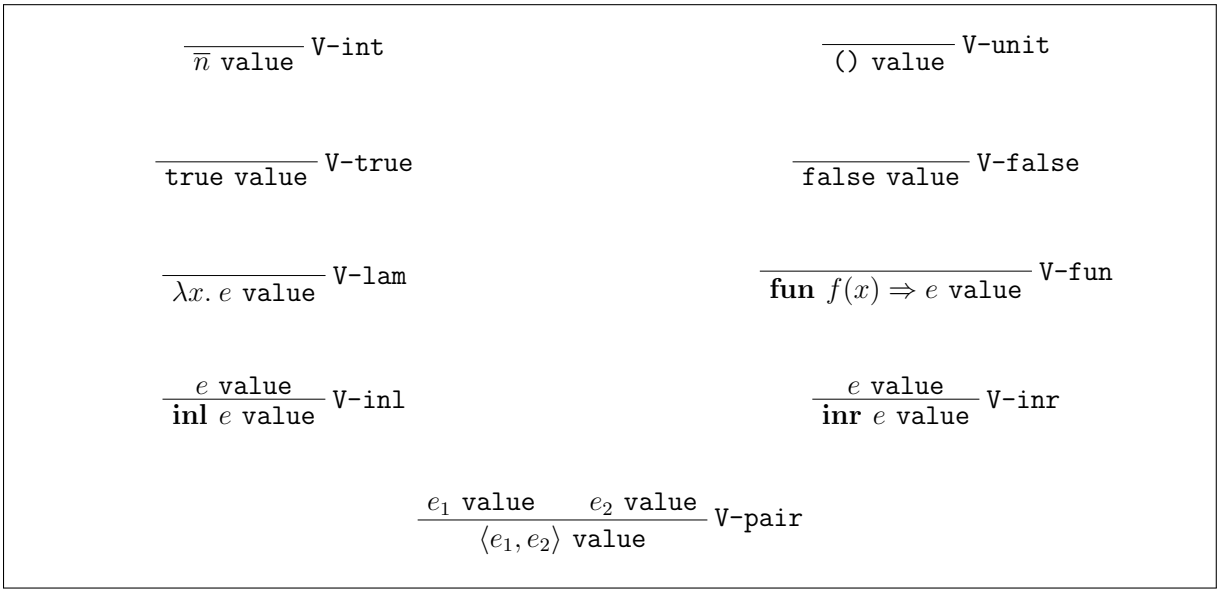

Figure 1.6: Inference rules for the judgment e `value`

To specify how programs execute, we define a small-step operational semantics for our language. This takes the form of a relation $e \hookrightarrow e'$ that describes a single computation step from an expression e to another expression e'. We adopt a left-to-right call-by-value strategy, where function arguments are fully evaluated before application, and evaluation proceeds from left to right. The rules defining this relation are given in Figure 1.7 in the next page. In these rules, we use v (and subscripted variants) to denote values. The symbols $\dot{*}$ and $\dot{\circ}$ range over syntactical operators, as introduced in the last section: $\dot{*} \in \{\dot{+}, \dot{-}, \dot{\times}\}$ for arithmetic and $\dot{\circ} \in \{\dot{=}, \dot{<}, \dot{>}\}$ for comparison. Their undotted counterparts denote the actual semantic operations.

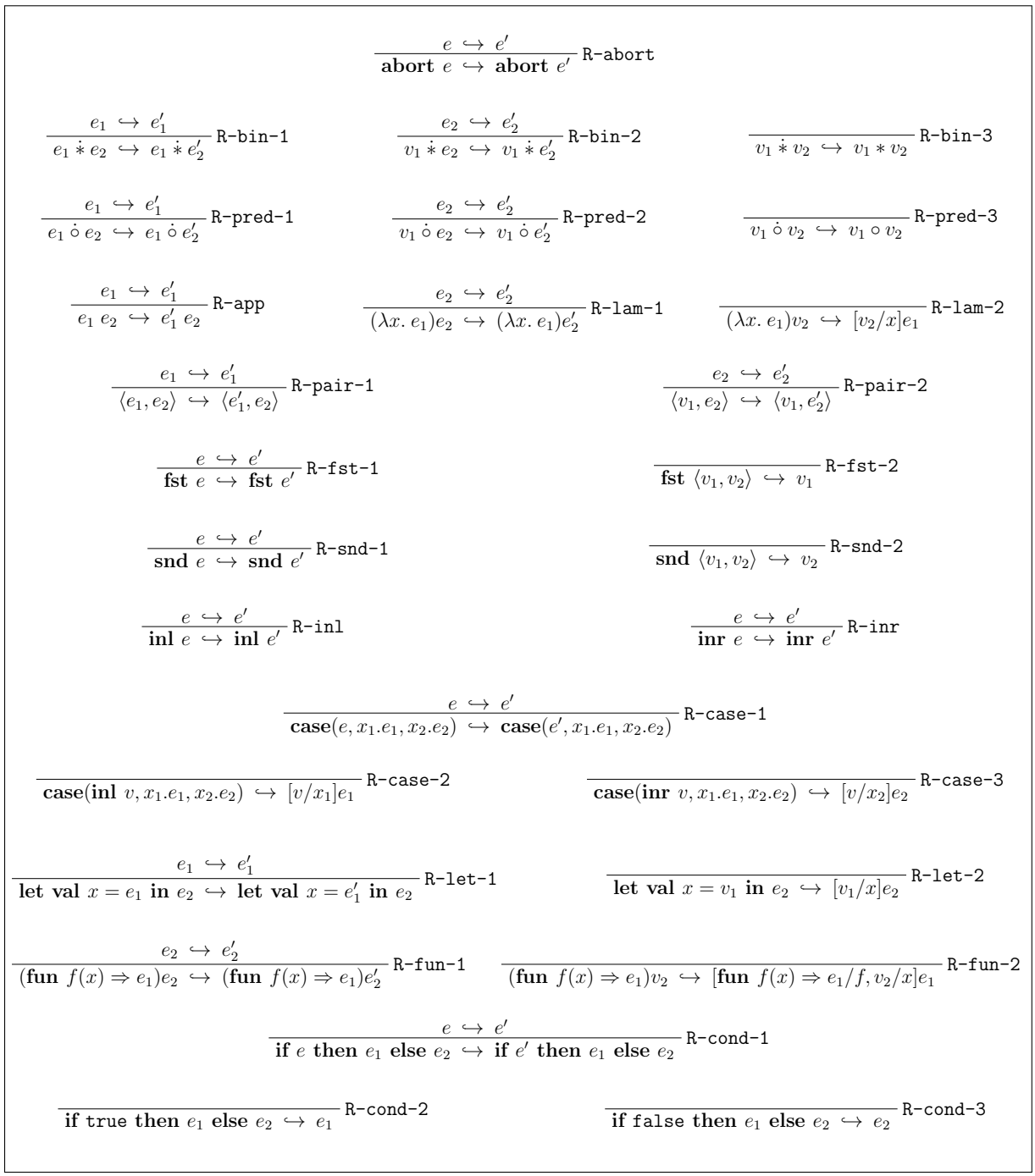

Figure 1.7: Small-step operational semantics

Typechecking serves a practical role: it allows us to verify that a program will behave well at runtime. The relationship between types and execution is formally captured by two key theorems: preservation and progress. Together, they establish that well-typed programs do not get stuck—they either are values or can take a reduction step.

Theorem 1.5.1 (Preservation)**.** If $\cdot \vdash e : A$ and $e \hookrightarrow e'$, then $\cdot \vdash e' : A$.

Proof. By rule induction on the derivation of $e \hookrightarrow e'$.

- **Case** `R-app`**.** We have $e = e_1\ e_2$ and $e_1 \hookrightarrow e_1'$, so $e' = e_1'\ e_2$. From the typing assumption, we know $\cdot \vdash e_1\ e_2 : B$, so we must have a derivation of the form:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \cdot \vdash e_1 : A \to B & \cdot \vdash e_2 : A\end{array}}{\cdot \vdash e_1\ e_2 : B}$$

  Applying the inductive hypothesis on $e_1 \hookrightarrow e_1'$, we get $\cdot \vdash e_1' : A \to B$. Then by applying $\to$`E` again, we conclude $\cdot \vdash e_1'\ e_2 : B$.

- **Case** `R-lam-1`**.** We have $e = (\lambda x.\, e'')\ e_2$ and $e_2 \hookrightarrow e_2'$, so $e' = (\lambda x.\, e'')\ e_2'$. From the typing assumption, we know $\cdot \vdash (\lambda x.\, e'')\ e_2 : B$, so we must have a derivation of the form:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \cdot \vdash \lambda x.\, e'' : A \to B & \cdot \vdash e_2 : A\end{array}}{\cdot \vdash (\lambda x.\, e'')\ e_2 : B}$$

  By the inductive hypothesis, $\cdot \vdash e_2' : A$, so by $\to$`E`, $\cdot \vdash (\lambda x.\, e'')\ e_2' : B$.

- **Case** `R-lam-2`**.** We have $e = (\lambda x.\, e'')\ v$ where v is a value, so $e' = [v/x]e''$. We know:

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ x{:}A \vdash e'' : B\end{array}}{\cdot \vdash \lambda x.\, e'' : A \to B} \qquad \cdot \vdash v : A}{\cdot \vdash (\lambda x.\, e'')\ v : B}$$

  By the Substitution Principle, we obtain: $\cdot \vdash [v/x]e'' : B$, which is the desired result.

- **Other cases.** These follow similarly by inversion of the typing derivation and reapplying the typing rules, using the Substitution Principle when necessary.

Thus, in all cases, if $\cdot \vdash e : A$ and $e \hookrightarrow e'$, then $\cdot \vdash e' : A$. □

Theorem 1.5.2 (Progress)**.** If $\cdot \vdash e : A$, then either e is a value, or there exists e' such that $e \hookrightarrow e'$.

Proof. By rule induction on the derivation of $\cdot \vdash e : A$.

- **Case** $\texttt{hyp}_x$**.** Impossible, since variables are never typable in the empty context.
- **Case** $\to\texttt{I}^x$**.** If $e = \lambda x.\, e'$, then e is a value by the rule `V-lam`.
- **Case** $\to\texttt{E}$**.** We have $e = e_1\ e_2$, with derivation:

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \cdot \vdash e_1 : A \to B & \cdot \vdash e_2 : A\end{array}}{\cdot \vdash e_1\ e_2 : B}$$

  By the inductive hypothesis, either e_1 is a value or there is some e_1' such that $e_1 \hookrightarrow e_1'$. Let us consider each subcase in turn.

  - If $e_1 \hookrightarrow e_1'$, then $e_1\ e_2 \hookrightarrow e_1'\ e_2$ by the `R-app` rule.
  - Suppose that e_1 is a value. Then e_1 must be of the form $\lambda x.\, e'$ for some expression e'. Applying the inductive hypothesis to e_2, we see that either e_2 is a value or there is some e_2' such that $e_2 \hookrightarrow e_2'$. In the former case, rule `R-lam-2` implies that $(\lambda x.\, e')e_2 \hookrightarrow [e_2/x]e'$, whereas in the latter case rule `R-lam-1` implies that $(\lambda x.\, e')e_2 \hookrightarrow (\lambda x.\, e')e_2'$. Hence, the desired conclusion holds in each case.

- **Other cases.** The result follows similarly by inspecting the typing rule and the corresponding evaluation rules. For example, when $e = \textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2$, we apply the inductive hypothesis to e_0 and consider two subcases:

  - If e_0 is a vaue, then it must be either `true` or `false`. If $e = \textbf{if}\ \texttt{true}\ \textbf{then}\ e_1\ \textbf{else}\ e_2$, then $e \hookrightarrow e_1$ by the `R-cond-2` rule. Similarly, if $e = \textbf{if}\ \texttt{false}\ \textbf{then}\ e_1\ \textbf{else}\ e_2$, then $e \hookrightarrow e_2$ by `R-cond-3`.
  - If e_0 is not a value, there is some e_0' such that $e_0 \hookrightarrow e_0'$, so $e \hookrightarrow \textbf{if}\ e_0'\ \textbf{then}\ e_1\ \textbf{else}\ e_2$ by the rule `R-cond-1`.

  A similar reasoning applies to let-bindings, pairs, case analysis, etc.

Thus, in all cases, either e is a value or $e \hookrightarrow e'$ for some e'. □

1.5.2 Implementing an Interpreter

In addition to the typechecker, we have also implemented an interpreter for the language, which evaluates expressions using the small-step operational semantics defined earlier. As before, to describe its implementation we restrict our attention to the simply-typed lambda calculus with booleans and integers.

The core of the interpreter consists of a function `eval` that performs a single reduction step according to the rules in Figure 1.7. A helper function `fullEval` applies `eval` repeatedly until a value is reached. To determine when evaluation should stop, we use the predicate `isValue`, defined in accordance with the value judgments presented in Figure 1.6.

```
isValue :: Term -> Bool
isValue Unit         = True -- V-unit
isValue (LitBool _)  = True -- V-true, V-false
isValue (LitInt _)   = True -- V-int
isValue (Lam _ _)    = True -- V-lam
isValue _            = False

eval :: Term -> Term
-- R-lam-2
eval (App (Lam x e1) v2) | isValue v2 =
subst x v2 e1

-- R-app, R-lam-1
eval (App e1 e2)
  | not (isValue e1) = App (eval e1) e2
  | otherwise        = App e1 (eval e2)

-- In other cases, the argument must be a value
eval t = t

fullEval :: Term -> Term
fullEval t =
  let t' = eval t
  in if t == t' then t else fullEval t'
```

Code Snippet 1.4: Implementation of the core operational semantics rules

2

Intuitionistic Modal Logic and Its Type-Theoretic Interpretation

In the previous chapter, we explored the Curry–Howard correspondence between intuitionistic propositional logic and the simply-typed lambda calculus. A natural question now arises: can this correspondence be extended to other logical systems, particularly those that reason about modalities such as necessity and possibility?

This chapter addresses that question by investigating intuitionistic modal logic, a system that introduces two modal operators—$\Box$ for necessity and $\Diamond$ for possibility. These modalities allow us to reason not just about what is true, but about what must be true or might be true, often interpreted in terms of multiple worlds or stages of computation.

2.1 Intuitionistic Modal Logic

"... there is no one fundamental logical notion of necessity, nor consequently of possibility. If this conclusion is valid, the subject of modality ought to be banished from logic, since propositions are simply true or false."

—Russell (1905)

"One often hears that modal (or some other) logic is pointless because it can be translated into some simpler language in a first-order way. Take no notice of such arguments ... What is essential is to single out important concepts and to investigate their properties."

—Scott (1970)

(Both quotes as presented by de Paiva (2015))

To explain the necessity and possibility modalities, we follow the approach of Pfenning and Davies (2001), who offer a foundation for intuitionistic modal logic grounded in Martin-Löf's methodology of distinguishing judgments from propositions—an approach discussed in the previous chapter.

We begin with the necessity modality, $\Box$, which expresses that a proposition holds universally, regardless of context. In a modal interpretation involving multiple worlds, $\Box A$

asserts that A is true in every world. In a proof-theoretic setting, the notion of a "world" is captured by the current set of assumptions; so a necessary proposition must hold under any set of assumptions; in particular, under no assumptions at all.

This perspective leads us to the Kantian notion of a *categorical judgment*: a judgment that does not rely on any truth hypotheses. It is a special case of a hypothetical judgment—one with an empty context of assumptions. To formalize this, we introduce a new judgment A `valid`, which expresses that A `true` holds without depending on any assumptions. Formally, we have:

- If $\cdot \vdash A$ `true`, then A `valid`.
- If A `valid`, then $\Gamma \vdash A$ `true` for any context Γ.

We now allow hypotheses of the form A `valid` to appear alongside ordinary assumptions in hypothetical judgments, writing them as:

$$u_1{::}B_1\ \texttt{valid}, \ldots, u_m{::}B_m\ \texttt{valid}; x_1{:}A_1\ \texttt{true}, \ldots, x_n{:}A_n\ \texttt{true} \vdash A\ \texttt{true},$$

where the u_i and x_j are distinct labels. For brevity, we write this more compactly as:

$$\Delta; \Gamma \vdash A\ \texttt{true},$$

where Δ contains valid assumptions and Γ contains ordinary assumptions.

To define the necessity modality, we now *internalize* the categorical judgment as a proposition. This yields the following introduction rule:

$$\frac{\Delta; \cdot \vdash A\ \texttt{true}}{\Delta; \Gamma \vdash \Box A\ \texttt{true}}\ \Box\mathtt{I}$$

The corresponding elimination rule is:

$$\frac{\Delta; \Gamma \vdash \Box A\ \texttt{true} \qquad \Delta, u{::}A\ \texttt{valid}; \Gamma \vdash B\ \texttt{true}}{\Delta; \Gamma \vdash B\ \texttt{true}}\ \Box\mathtt{E}^u$$

As in the previous chapter, we show that these rules are in harmony by verifying local soundness and local completeness. Local soundness is demonstrated by the following local reduction, which eliminates an unnecessary detour:

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \cdot \vdash A\ \texttt{true}\end{array}}{\Delta; \Gamma \vdash \Box A\ \texttt{true}}\ \Box\mathtt{I} \qquad \begin{array}{c}\mathcal{E}\\ \Delta, u{::}A\ \texttt{valid}; \Gamma \vdash B\ \texttt{true}\end{array}}{\Delta; \Gamma \vdash B\ \texttt{true}}\ \Box\mathtt{E}^u \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{E}'\\ \Delta; \Gamma \vdash B\ \texttt{true}\end{array}$$

where $\mathcal{E}'$ is obtained from $\mathcal{E}$ by substituting $\mathcal{D}$ for uses of the hypothesis that A is valid.

Local completeness, on the other hand, is witnessed by the following local expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \Box A\ \mathtt{true}\end{array} \quad \Longrightarrow_E \quad \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash \Box A\ \mathtt{true}\end{array} \qquad \dfrac{\dfrac{}{\Delta, u{::}A\ \mathtt{valid}; \cdot \vdash A\ \mathtt{true}}\,\mathtt{hyp}^*_u}{\Delta, u{::}A\ \mathtt{valid}; \Gamma \vdash \Box A\ \mathtt{true}}\,\Box\mathtt{I}}{\Delta; \Gamma \vdash \Box A\ \mathtt{true}}\,\Box\mathtt{E}^u$$

We now turn to the possibility modality, $\Diamond$. Intuitively, to say that a proposition A is possibly true means that there exists some world in which A holds. Unlike necessity, which demands that A be true in all conceivable contexts, possibility is satisfied by the existence of a single world where A is true.

We introduce a new judgment $A\ \mathtt{poss}$, which expresses that A is possibly true. Our hypothetical judgments now can also have the form

$$\Delta; \Gamma \vdash A\ \mathtt{poss}.$$

This judgment is defined by the following two principles:

- If $\Delta; \Gamma \vdash A\ \mathtt{true}$, then $\Delta; \Gamma \vdash A\ \mathtt{poss}$.
- If $\Delta; \Gamma \vdash A\ \mathtt{poss}$ and $\Delta; A\ \mathtt{true} \vdash C\ \mathtt{poss}$, then $\Delta; \Gamma \vdash C\ \mathtt{poss}$.

The first rule tells us that actual truth implies possible truth. The second expresses that possibility can be propagated: if A is possibly true and from the truth of A we can derive the possibility of C, then C is also possibly true.

We then internalize this judgment into a proposition $\Diamond A$, with the following natural deduction rules:

$$\frac{\Delta; \Gamma \vdash A\ \mathtt{poss}}{\Delta; \Gamma \vdash \Diamond A\ \mathtt{true}}\,\Diamond\mathtt{I} \qquad\qquad \frac{\Delta; \Gamma \vdash \Diamond A\ \mathtt{true} \qquad \Delta; x{:}A\ \mathtt{true} \vdash B\ \mathtt{poss}}{\Delta; \Gamma \vdash B\ \mathtt{poss}}\,\Diamond\mathtt{E}^x$$

As before, these rules satisfy local soundness and completeness. For local soundness, we have the following reduction:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash A\ \mathtt{true}\end{array}}{\Delta; \Gamma \vdash A\ \mathtt{poss}}\,\mathtt{coerce}}{\Delta; \Gamma \vdash \Diamond A\ \mathtt{true}}\,\Diamond\mathtt{I} \qquad \begin{array}{c}\mathcal{E}\\ \Delta; A\ \mathtt{true} \vdash B\ \mathtt{poss}\end{array}}{\Delta; \Gamma \vdash B\ \mathtt{poss}} \quad \Longrightarrow_R \quad \begin{array}{c}\mathcal{E}'\\ \Delta; \Gamma \vdash B\ \mathtt{poss}\end{array}$$

Local completeness is witnessed by the following local expansion:

$$\dfrac{\mathcal{D}}{\Gamma \vdash \Diamond A\ \mathtt{true}} \Longrightarrow_E \dfrac{\dfrac{\mathcal{D}}{\Delta;\Gamma \vdash \Diamond A\ \mathtt{true}} \quad \dfrac{\dfrac{}{\Delta; x{:}A\ \mathtt{true} \vdash A\ \mathtt{true}}\,\mathtt{hyp}_x}{\Delta; x{:}A\ \mathtt{true} \vdash A\ \mathtt{poss}}\,\mathtt{coerce}}{\dfrac{\Delta;\Gamma \vdash A\ \mathtt{poss}}{\Delta;\Gamma \vdash \Diamond A\ \mathtt{true}}\,\Diamond\mathtt{I}}\,\Diamond\mathtt{E}^x$$

Additionally, we allow the following rule, which accounts for the interaction between a valid hypothesis and a possible truth in the conclusion:

$$\dfrac{\Delta;\Gamma \vdash \Box A\ \mathtt{true} \qquad \Delta, u{::}A\ \mathtt{valid};\Gamma \vdash B\ \mathtt{poss}}{\Delta;\Gamma \vdash B\ \mathtt{poss}}\,\Box\mathtt{E}^u_p$$

Local soundness for this rule is witnessed by the following local reduction:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Delta;\cdot \vdash A\ \mathtt{true}}}{\Delta;\Gamma \vdash \Box A\ \mathtt{true}}\,\Box\mathtt{I} \qquad \mathcal{E}}{\Delta;\Gamma \vdash B\ \mathtt{poss}}\,\Box\mathtt{E}^u_p \Longrightarrow_R \qquad \dfrac{\mathcal{E}'}{\Delta;\Gamma \vdash B\ \mathtt{true}}$$

The full list of natural deduction rules is presented in Figure 2.1.

$$\dfrac{}{\Delta, u{::}A\ \mathtt{valid}, \Delta';\Gamma \vdash A\ \mathtt{true}}\,\mathtt{hyp}^*_u$$

$$\dfrac{\Delta;\cdot \vdash A\ \mathtt{true}}{\Delta;\Gamma \vdash \Box A\ \mathtt{true}}\,\Box\mathtt{I} \qquad \dfrac{\Delta;\Gamma \vdash \Box A\ \mathtt{true} \qquad \Delta, u{::}A\ \mathtt{valid};\Gamma \vdash B\ \mathtt{true}}{\Delta;\Gamma \vdash B\ \mathtt{true}}\,\Box\mathtt{E}^u$$

$$\dfrac{\Delta;\Gamma \vdash A\ \mathtt{true}}{\Delta;\Gamma \vdash A\ \mathtt{poss}}\,\mathtt{coerce}$$

$$\dfrac{\Delta;\Gamma \vdash \Box A\ \mathtt{true} \qquad \Delta, u{::}A\ \mathtt{valid};\Gamma \vdash B\ \mathtt{poss}}{\Delta;\Gamma \vdash B\ \mathtt{poss}}\,\Box\mathtt{E}^u_p$$

$$\dfrac{\Delta;\Gamma \vdash A\ \mathtt{poss}}{\Delta;\Gamma \vdash \Diamond A\ \mathtt{true}}\,\Diamond\mathtt{I} \qquad \dfrac{\Delta;\Gamma \vdash \Diamond A\ \mathtt{true} \qquad \Delta; x{:}A\ \mathtt{true} \vdash B\ \mathtt{poss}}{\Delta;\Gamma \vdash B\ \mathtt{poss}}\,\Diamond\mathtt{E}^x$$

Figure 2.1: Natural deduction rules for necessity and possibility

Notice that we have two structural rules: $\mathtt{hyp}^*_u$ and $\mathtt{coerce}$, besides five logical rules. As we did in the previous chapter, we annotate these with proof terms. For this purpose, we introduce a new class of proof terms, which we call *computations*. Recall that the judgment $e : A$ indicates that expression e is a proof term showing evidence for the judgment $A\ \mathtt{true}$,

or, equivalently, that e is a program having type A, according to the rules presented in the previous chapter. Since we now have a new judgment A `poss`, we introduce a new judgment $c \div A$, which indicates that computation c is a proof term showing evidence for the judgment A `poss`. The annotated rules are presented in Figure 2.2.

$$\frac{}{\Delta, u{::}A, \Delta'; \Gamma \vdash u : A}\ \mathtt{hyp}^*_u$$

$$\frac{\Delta; \cdot \vdash e : A}{\Delta; \Gamma \vdash \mathbf{box}\ e : \Box A}\ \Box\mathtt{I} \qquad \frac{\Delta; \Gamma \vdash e_1 : \Box A \qquad \Delta, u{::}A; \Gamma \vdash e_2 : B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 : B}\ \Box\mathtt{E}^u$$

$$\frac{\Delta; \Gamma \vdash e : A}{\Delta; \Gamma \vdash \mathbf{ret}\ e \div A}\ \mathtt{coerce}$$

$$\frac{\Delta; \Gamma \vdash e : \Box A \qquad \Delta, u{::}A; \Gamma \vdash c \div B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e\ \mathbf{in}\ c \div B}\ \Box\mathtt{E}^u_p$$

$$\frac{\Delta; \Gamma \vdash c \div A}{\Delta; \Gamma \vdash \mathbf{do}\ c : \Diamond A}\ \Diamond\mathtt{I} \qquad \frac{\Delta; \Gamma \vdash e : \Diamond A \qquad \Delta; x{:}A \vdash c \div B}{\Delta; \Gamma \vdash x \leftarrow e; c \div B}\ \Diamond\mathtt{E}^x$$

Figure 2.2: Proof term annotations for necessity and possibility

Our proof terms follow those of Pfenning and Davies (2001), except for the terms related to possibility, where we adopt a Haskell-inspired notation to emphasize the computational structure of modal proofs: the construct **ret** e returns a value, **do** c initiates a sequence of computations, and $x \leftarrow e; c$ expresses sequential composition with binding. Let us now look at some examples.

Example 2.1.1.

- $\vdash \lambda x.\, \lambda y.\, \mathbf{let\ box}\ u = x\ \mathbf{in\ let\ box}\ w = y\ \mathbf{in}\ (u\ w) : \Box(A \to B) \to (\Box A \to \Box B)$.
- $\vdash \lambda x.\, \mathbf{let\ box}\ u = x\ \mathbf{in}\ u : \Box A \to A$.
- $\vdash \lambda x.\, \mathbf{let\ box}\ u = x\ \mathbf{in\ box\ box}\ u : \Box A \to \Box\Box A$.
- $\vdash \lambda x.\, \mathbf{do\ ret}\ x : A \to \Diamond A$.
- $\vdash \lambda x.\, \mathbf{do}\ y \leftarrow z; z \leftarrow y; \mathbf{ret}\ z : \Diamond\Diamond A \to \Diamond A$.
- $\vdash \lambda x.\, \lambda y.\, \mathbf{let\ box}\ u = x\ \mathbf{in}\ (\mathbf{do}\ z \leftarrow y; \mathbf{ret}\ (u\ z)) : \Box(A \to B) \to (\Diamond A \to \Diamond B)$.

Since we now have a new kind of variable, we introduce a corresponding substitution operation, called a *modal substitution*. We write $[\![e/u]\!]e'$ for the result of substituting the term e for all free occurrences of the modal variable u in e'. Together with the definition of the set of free modal variables of a term, $\mathtt{FMV}(\cdot)$, this operation is specified below. We present only the most relevant cases; the remaining ones are defined analogously.

$$
\begin{array}{ll}
\mathtt{FMV}(x) & = \varnothing \\
\mathtt{FMV}(\lambda x.\, e) & = \mathtt{FMV}(e) \\
\mathtt{FMV}(e_1\ e_2) & = \mathtt{FMV}(e_1) \cup \mathtt{FMV}(e_2) \\
\mathtt{FMV}(u) & = \{u\} \\
\mathtt{FMV}(\mathbf{box}\ e) & = \mathtt{FMV}(e) \\
\mathtt{FMV}(\mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2) & = \mathtt{FMV}(e_1) \cup (\mathtt{FMV}(e_2 \setminus \{u\})) \\
\mathtt{FMV}(\mathbf{ret}\ e) & = \mathtt{FMV}(e) \\
\mathtt{FMV}(\mathbf{do}\ c) & = \mathtt{FMV}(c) \\
\mathtt{FMV}(x \leftarrow e; c) & = \mathtt{FMV}(e) \cup \mathtt{FMV}(c)
\end{array}
$$

$$
\begin{array}{ll}
[\![e/u]\!]x & = x \\
[\![e/u]\!]\lambda x.\, e' & = \lambda x.\, [\![e/u]\!]e' \\
[\![e/u]\!](e_1\ e_2) & = ([\![e/u]\!]e_1)\ ([\![e/u]\!]e_2) \\
[\![e/u]\!]u & = e \\
[\![e/u]\!]v & = v \quad \text{provided that } u \neq v \\
[\![e/u]\!]\mathbf{box}\ e' & = \mathbf{box}\ [\![e/u]\!]e' \\
[\![e/u]\!]\mathbf{let\ box}\ v = e_1\ \mathbf{in}\ e_2 & = \mathbf{let\ box}\ v = [\![e/u]\!]e_1\ \mathbf{in}\ [\![e/u]\!]e_2 \quad \text{provided that } v \notin \mathtt{FMV}(e_2), u \neq v \\
[\![e/u]\!]\mathbf{ret}\ e' & = \mathbf{ret}\ [\![e/u]\!]e' \\
[\![e/x]\!]\mathbf{do}\ e' & = \mathbf{do}\ [\![e/u]\!]e' \\
[\![e/x]\!]x \leftarrow e'; c & = x \leftarrow [\![e/u]\!]e'; [\![e/u]\!]c
\end{array}
$$

As in the previous chapter, the following result ensures that the modal substitution operation is well-behaved with respect to typing.

Theorem 2.1.2 (Substitution Lemma for Validity)**.** If $\Delta; \cdot \vdash e' : A$ and $\Delta, u{::}A; \Gamma \vdash e : B$, then $\Delta; \Gamma \vdash [\![e'/u]\!]e : B$.

Proof. By rule induction on the derivation of the judgment $\Delta, u{::}A; \Gamma \vdash e : B$. □

We also need a new substitution operation, $\langle\langle\, c'/x \,\rangle\rangle c$, which denotes the result of substituting a computation c' for the free occurrences of a variable x in another computation c. We define this by induction on c' as follows:

$$
\begin{array}{ll}
\langle\langle\, \mathbf{ret}\ e/x \,\rangle\rangle c & = [e/x]c \\
\langle\langle\, y \leftarrow e; c'/x \,\rangle\rangle c & = y \leftarrow e; \langle\langle\, c'/x \,\rangle\rangle c \\
\langle\langle\, \mathbf{let\ box}\ u = e\ \mathbf{in}\ c'/x \,\rangle\rangle c & = \mathbf{let\ box}\ u = e\ \mathbf{in}\ \langle\langle\, c'/x \,\rangle\rangle c
\end{array}
$$

We can now express the local reductions and expansions we have seen in a more compact way. For necessity, we have the following:

$$
\begin{array}{lll}
\Delta; \Gamma \vdash \mathbf{let\ box}\ u = \mathbf{box}\, e'\ \mathbf{in}\ e : A & \Longrightarrow_R & \Delta; \Gamma \vdash [\![e'/u]\!] e : A \\
\Delta; \Gamma \vdash e : \Box A & \Longrightarrow_E & \Delta; \Gamma \vdash \mathbf{let\ box}\ u = e\ \mathbf{in\ box}\, u : \Box A \\
\Delta; \Gamma \vdash \mathbf{let\ box}\ u = \mathbf{box}\, e'\ \mathbf{in}\ c \div A & \Longrightarrow_R & \Delta; \Gamma \vdash [\![e'/u]\!] c \div A
\end{array}
$$

Whereas for possibility we have:

$$
\begin{array}{lll}
\Delta; \Gamma \vdash x \leftarrow \mathbf{do}\ c'; c \div B & \Longrightarrow_R & \Delta; \Gamma \vdash \langle\langle\, c'/x \,\rangle\rangle c \div B \\
\Delta; \Gamma \vdash e : \Diamond A & \Longrightarrow_E & \Delta; \Gamma \vdash \mathbf{do}\ (x \leftarrow e; \mathbf{ret}\ x) : \Diamond A
\end{array}
$$

2.2 Typechecking Programs

We now formulate the rules in Figure 2.2 in a bidirectional way. This is shown in Figure 2.3, where we use an additional judgment $c \stackrel{\bullet}{\Leftarrow} A$, where c is an annotated computation, to mirror the judgment $|c| \div A$.

$$\frac{}{\Delta, u{::}A, \Delta'; \Gamma \vdash u \Rightarrow A}\ \texttt{T-mvar}$$

$$\frac{\Delta; \cdot \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{box}\, e \Leftarrow \Box A}\ \texttt{T-box} \qquad \frac{\Delta; \Gamma \vdash e_1 \Rightarrow \Box A \qquad \Delta, u{::}A; \Gamma \vdash e_2 \Leftarrow B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \Leftarrow B}\ \texttt{T-letbox}$$

$$\frac{\Delta; \Gamma \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{ret}\ e \stackrel{\bullet}{\Leftarrow} A}\ \texttt{T-coerce}$$

$$\frac{\Delta; \Gamma \vdash e \Rightarrow \Box A \qquad \Delta, u{::}A; \Gamma \vdash c \stackrel{\bullet}{\Leftarrow} B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e\ \mathbf{in}\ c \stackrel{\bullet}{\Leftarrow} B}\ \texttt{T-letbox-p}$$

$$\frac{\Delta; \Gamma \vdash c \stackrel{\bullet}{\Leftarrow} A}{\Delta; \Gamma \vdash \mathbf{do}\ c \Leftarrow \Diamond A}\ \texttt{T-do} \qquad \frac{\Delta; \Gamma \vdash e \Rightarrow \Diamond A \qquad \Delta; x{:}A \vdash c \stackrel{\bullet}{\Leftarrow} B}{\Delta; \Gamma \vdash x \leftarrow e; c \stackrel{\bullet}{\Leftarrow} B}\ \texttt{T-seq}$$

Figure 2.3: Bidirectional typechecking rules for modal terms

The following two theorems establish the correspondence between our augmented typing an bidirectional rules.

Theorem 2.2.1 (Soundness)**.** The following hold:

(i) If $\Delta; \Gamma \vdash e \Leftarrow A$, then $\Delta; \Gamma \vdash |e| : A$.

(ii) If $\Delta; \Gamma \vdash e \Rightarrow A$, then $\Delta; \Gamma \vdash |e| : A$.

(iii) If $\Delta; \Gamma \vdash c \overset{\bullet}{\Leftarrow} A$, then $\Delta; \Gamma \vdash |c| \div A$.

Proof. The proof proceeds by mutual rule induction on the derivation of the judgments $\Delta; \Gamma \vdash e \Leftarrow A$, $\Delta; \Gamma \vdash e \Rightarrow A$, and $\Delta; \Gamma \vdash c \overset{\bullet}{\Leftarrow} A$.

(i)

- **Case** `T-box`**.** In this case, there is a derivation of the form

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \cdot \vdash e \Leftarrow A\end{array}}{\Delta; \Gamma \vdash \mathbf{box}\, e \Leftarrow A}$$

  Applying the inductive hypothesis on the premise, we deduce that $\Delta; \Gamma \vdash |e| : A$. Then, by rule $\Box\mathtt{I}$ and since $|\mathbf{box}\, e| = \mathbf{box}\, |e|$, it follows that $\Delta; \Gamma \vdash |\mathbf{box}\, e| : \Box A$.

- **Case** `T-letbox`**.** The corresponding derivation here has the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash e_1 \Rightarrow \Box A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, u{::}A; \Gamma \vdash e_2 \Leftarrow B\end{array}}{\Delta; \Gamma \vdash \mathbf{let}\ \mathbf{box}\ u = e_1\ \mathbf{in}\ e_2 \Leftarrow B}$$

  Using part (ii) of this theorem on the first premise and the inductive hypothesis on the second, we obtain the necessary hypotheses to apply the rule $\Box\mathtt{E}^u$ and deduce that $\Delta; \Gamma \vdash |\mathbf{let}\ \mathbf{box}\ u = e_1\ \mathbf{in}\ e_2| : B$, as desired.

- **Case** `T-do`**.** Here, the corresponding derivation tree has the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash c \overset{\bullet}{\Leftarrow} A\end{array}}{\Delta; \Gamma \vdash \mathbf{do}\ c \Leftarrow \Diamond A}\ \mathtt{T\text{-}do}$$

  Applying part (iii) of this theorem to the premise yields the necessary hypothesis to use the rule $\Diamond\mathtt{I}$ and infer that $\Delta; \Gamma \vdash |\mathbf{do}\ c| : \Diamond A$.

(ii)

- **Case** `T-mvar`**.** Here, we have the following derivation:

$$\dfrac{}{\Delta, u{::}A; \Gamma \vdash u \Rightarrow A}$$

  As u is in the modal context, we can apply the rule $\mathtt{hyp}^*_u$ to deduce the desired result.

(iii)

- **Case `T-coerce`.**

$$\dfrac{\begin{array}{c}\vdots\\ \Delta;\Gamma \vdash e \Leftarrow A\end{array}}{\Delta;\Gamma \vdash \mathbf{ret}\ e \overset{\bullet}{\Leftarrow} A}\ \texttt{T-coerce}$$

Applying part (i) of this theorem to the premise yields $\Delta;\Gamma \vdash |e| : A$, so by rule `coerce`, we obtain $\Delta;\Gamma \vdash |\mathbf{ret}\ e| \div A$, as desired.

- **Case `T-letbox-p`.** The corresponding derivation here has the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta;\Gamma \vdash e_1 \Rightarrow \Box A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, u{::}A;\Gamma \vdash e_2 \overset{\bullet}{\Leftarrow} B\end{array}}{\Delta;\Gamma \vdash \mathbf{let}\ \mathbf{box}\ u = e_1\ \mathbf{in}\ e_2 \overset{\bullet}{\Leftarrow} B}$$

Using part (ii) of this theorem on the first premise and the inductive hypothesis on the second, we obtain the necessary hypotheses to apply the rule $\Box\mathtt{E}^u_p$ and deduce that $\Delta;\Gamma \vdash |\mathbf{let}\ \mathbf{box}\ u = e_1\ \mathbf{in}\ e_2| \div B$, as desired.

- **Case `T-seq`.** In this case, the corresponding derivation has the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta;\Gamma \vdash e \Rightarrow \Diamond A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta; x{:}A \vdash c \overset{\bullet}{\Leftarrow} B\end{array}}{\Delta;\Gamma \vdash x \leftarrow e; c \overset{\bullet}{\Leftarrow} B}\ \texttt{T-seq}$$

Applying part (i) of this theorem to the first premise and the inductive hypothesis to the second provides the necessary hypotheses to apply the rule $\Diamond\mathtt{E}^x$ and deduce that $\Delta;\Gamma \vdash |x \leftarrow e; c| \div B$, as desired.

□

Theorem 2.2.2 (Completeness)**.** The following hold:

(i) If $\Delta;\Gamma \vdash e : A$, there exists some e' such that $\Delta;\Gamma \vdash e' \Leftarrow A$ and $|e'| = e$.

(ii) If $\Delta;\Gamma \vdash c \div A$, there exists some c' such that $\Delta;\Gamma \vdash c' \overset{\bullet}{\Leftarrow} A$ and $|c'| = c$.

Proof.

(i) By rule induction on the derivation of the judgment $\Delta;\Gamma \vdash e : A$.

- **Case $\mathtt{hyp}^*_u$.** In this case, we have a derivation of the form:

$$\dfrac{}{\Delta, u{::}A, \Delta';\Gamma \vdash u : A}$$

We can then form the following derivation tree:

$$\dfrac{\dfrac{}{\Delta, u{::}A, \Delta'; \Gamma \vdash u \Rightarrow A}\,\texttt{T-mvar}}{\Delta, u{::}A, \Delta'; \Gamma \vdash u \Leftarrow A}\,\texttt{T-sub}$$

Hence, letting $e' = u$ works in this case.

- **Case** $\Box\texttt{I}$**.** Here, we have a derivation of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \cdot \vdash e : A\end{array}}{\Delta; \Gamma \vdash \mathbf{box}\, e : \Box A}$$

By applying the inductive hypothesis to the premise, there exists some e' such that $\Delta; \cdot \vdash e' \Leftarrow A$. Application of the rule `T-box` immediately yields that $\Delta; \cdot \vdash \mathbf{box}\, e' \Leftarrow A$, so our annotated expression in this case is $\mathbf{box}\, e'$.

- **Case** $\Box\texttt{E}^u$**.** In this case, there is a derivation tree of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash e_1 : \Box A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, u{::}A; \Gamma \vdash e_2 : B\end{array}}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 : B}$$

The inductive hypothesis implies the existence of e_1' and e_2' such that $|e_1'| = e_1$, $|e_2'| = e_2$, and the existence of the following derivations:

$$\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e_1' \Leftarrow \Box A\end{array} \qquad\qquad \begin{array}{c}\mathcal{E}\\ \Delta, u{::}A; \Gamma \vdash e_2' \Leftarrow B\end{array}$$

We can then construct the following derivation tree:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e_1' \Leftarrow \Box A\end{array}}{\Delta; \Gamma \vdash (e_1' : \Box A) \Rightarrow \Box A}\,\texttt{T-anno} \qquad \begin{array}{c}\mathcal{E}\\ \Delta, u{::}A; \Gamma \vdash e_2' \Leftarrow B\end{array}}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = (e_1' : \Box A)\ \mathbf{in}\ e_2 \Leftarrow B}\,\texttt{T-letbox}$$

- **Case** $\Diamond\texttt{I}$**.** In this case, we have a derivation of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash c \div A\end{array}}{\Delta; \Gamma \vdash \mathbf{do}\ c : \Diamond A}$$

Applying part (ii) of this theorem to the premise, we get that there is some c' such that $\Delta; \Gamma \vdash c' \overset{\bullet}{\Leftarrow} A$ and $|c'| = c$. Applying the rule `T-do` yields the desired result.

(ii) By rule induction on the derivation of the judgment $\Delta; \Gamma \vdash c \div A$.

- **Case** `coerce`**.** Here, we have a derivation tree of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash c : A\end{array}}{\Delta; \Gamma \vdash \mathbf{ret}\ c \div A}$$

Applying part (i) of this theorem to the premise allows us to deduce that there is some annotated version c' of c such that $\Delta; \Gamma \vdash c' \Leftarrow A$. Now, using the rule `T-coerce`, we see that $\Delta; \Gamma \vdash \mathbf{do}\ c' \overset{\bullet}{\Leftarrow} A$, as desired.

- **Case $\Box\mathrm{E}^u_p$.** In this case, there is a derivation tree of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash e_1 : \Box A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, u{::}A; \Gamma \vdash e_2 \div B\end{array}}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \div B}$$

Using part (i) of this theorem and the inductive hypothesis, we deduce that there are e'_1 and e'_2 such that $|e'_1| = e_1$, $|e'_2| = e_2$, and the existence of the following derivations:

$$\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e'_1 \Leftarrow \Box A\end{array} \qquad\qquad \begin{array}{c}\mathcal{E}\\ \Delta, u{::}A; \Gamma \vdash e'_2 \overset{\bullet}{\Leftarrow} B\end{array}$$

We can then construct the following derivation tree:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e'_1 \Leftarrow \Box A\end{array}}{\Delta; \Gamma \vdash (e'_1 : \Box A) \Rightarrow \Box A}\,\texttt{T-anno} \qquad \begin{array}{c}\mathcal{E}\\ \Delta, u{::}A; \Gamma \vdash e'_2 \overset{\bullet}{\Leftarrow} B\end{array}}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = (e'_1 : \Box A)\ \mathbf{in}\ e_2 \overset{\bullet}{\Leftarrow} B}\,\texttt{T-letbox-p}$$

- **Case $\Diamond\mathrm{E}^x$.** Here, there is a derivation tree of the form:

$$\dfrac{\begin{array}{c}\vdots\\ \Delta; \Gamma \vdash e : \Diamond A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta; x{:}A \vdash c \div B\end{array}}{\Delta; \Gamma \vdash x \leftarrow e; c \div B}$$

Using part (i) of this theorem on the first premise, and the inductive hypothesis on the second, we can deduce the existence of the following derivation trees:

$$\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e \Leftarrow \Diamond A\end{array} \qquad\qquad \begin{array}{c}\mathcal{E}\\ \Delta; x{:}A \vdash c \overset{\bullet}{\Leftarrow} B\end{array}$$

We can then construct the following derivation tree:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta; \Gamma \vdash e \Leftarrow \Diamond A\end{array}}{\Delta; \Gamma \vdash (e : \Diamond A) \Rightarrow \Diamond A}\,\texttt{T-anno} \qquad \begin{array}{c}\mathcal{E}\\ \Delta; x{:}A \vdash c \overset{\bullet}{\Leftarrow} B\end{array}}{\Delta; \Gamma \vdash x \leftarrow e; c \overset{\bullet}{\Leftarrow} B}\,\texttt{T-seq}$$

□

Our typechecker implementation can be readily extended to support the augmented language. The details for handling the modal constructs will be discussed in the next chapter, where we turn to contextual modalities, of which simple modalities are a special case.

2.3 Running Programs

To execute programs in the extended language, we first add new inference rules for the judgment e `value`.

$$\frac{}{\mathbf{box}\ e\ \mathtt{value}}\ \mathtt{V\text{-}box}$$

$$\frac{v\ \mathtt{value}}{\mathbf{ret}\ v\ \mathtt{value}}\ \mathtt{V\text{-}ret} \qquad \frac{}{\mathbf{do}\ c\ \mathtt{value}}\ \mathtt{V\text{-}do}$$

Figure 2.4: Additional inference rule for the judgment e `value`

Our operational semantics requires the addition of the following rules:

$$\frac{e \hookrightarrow e'}{\mathbf{ret}\ e \hookrightarrow \mathbf{ret}\ e'}\ \mathtt{R\text{-}ret}$$

$$\frac{e \hookrightarrow e'}{x \leftarrow e; c \hookrightarrow x \leftarrow e'; c}\ \mathtt{R\text{-}seq\text{-}1} \qquad \frac{c_1 \hookrightarrow c_1'}{(x \leftarrow \mathbf{do}\ c_1); c_2 \hookrightarrow (x \leftarrow \mathbf{do}\ c_1'); c_2}\ \mathtt{R\text{-}seq\text{-}2}$$

$$\frac{}{x \leftarrow \mathbf{do}\ \mathbf{ret}\ v; c_2 \hookrightarrow \langle\langle \mathbf{ret}\ v/x\rangle\rangle c_2}\ \mathtt{R\text{-}seq\text{-}3}$$

$$\frac{e_1 \hookrightarrow e_1'}{\mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \hookrightarrow \mathbf{let\ box}\ u = e_1'\ \mathbf{in}\ e_2}\ \mathtt{R\text{-}letbox\text{-}1} \qquad \frac{}{\mathbf{let\ box}\ u = \mathbf{box}\, e\ \mathbf{in}\ e_2 \hookrightarrow [\![e/u]\!]e_2}\ \mathtt{R\text{-}letbox\text{-}2}$$

Figure 2.5: Operational semantics rules for modal constructs

As before, we can prove that the properties of preservation and progress still hold.

Theorem 2.3.1 (Preservation)**.** The following hold:

(i) If $\cdot \vdash e : A$ and $e \hookrightarrow e'$, then $\cdot \vdash e' : A$.

(ii) If $\cdot \vdash c \div A$ and $c \hookrightarrow c'$, then $\cdot \vdash c' \div A$.

Proof. (i) By rule induction on the derivation of the judgment $e \hookrightarrow e'$.

- **Case** `R-letbox-1`**.** Here, we have must have a derivation of the form:

$$\frac{\begin{array}{c}\vdots\\ e_1 \hookrightarrow e_1'\end{array}}{\textbf{let box } u = e_1 \textbf{ in } e_2 \hookrightarrow \textbf{let box } u = e_1' \textbf{ in } e_2}$$

Additionally, there must be a typing derivation of the form:

$$\frac{\begin{array}{c}\vdots\\ \vdash e_1 : \Box A\end{array} \qquad \begin{array}{c}\vdots\\ u{::}A; \cdot \vdash e_2 : B\end{array}}{\vdash \textbf{let box } u = e_1 \textbf{ in } e_2 : B}$$

Applying the inductive hypothesis to the premise in the first derivation allows us to deduce that $\vdash e_1' : \Box A$. This provides all the necessary premises to deduce that $\vdash \textbf{let box } u = e_1' \textbf{ in } e_2 : B$, as required.

- **Case** `R-letbox-2`**.** In this case, our derivation has the form

$$\frac{}{\textbf{let box } u = \textbf{box } e \textbf{ in } e_2 \hookrightarrow [\![e/u]\!]e_2}$$

By hypothesis, we also have a derivation of the form:

$$\frac{\dfrac{\begin{array}{c}\vdots\\ \vdash e : A\end{array}}{\vdash \textbf{box } e : \Box A} \qquad \begin{array}{c}\vdots\\ u{::}A; \cdot \vdash e_2 : B\end{array}}{\vdash \textbf{let box } u = e_1 \textbf{ in } e_2 : B}$$

By the Substitution Lemma for Validity (Theorem 2.1.2), we deduce that $\vdash [\![e/u]\!]e_2 : B$.

(ii) By rule induction on the derivation of the judgment $c \hookrightarrow c'$.

- **Case** `R-ret`**.** In this case, our derivation has the form

$$\frac{\begin{array}{c}\vdots\\ e \hookrightarrow e'\end{array}}{\textbf{ret } e \hookrightarrow \textbf{ret } e'}$$

By hypothesis, we have a derivation of the form

$$\frac{\begin{array}{c}\vdots\\ \vdash e : A\end{array}}{\vdash \textbf{ret } e \div A}$$

We can then use the first part of this theorem to deduce that $\vdash e' : A$, and use our `coerce` rule to deduce that $\vdash \textbf{ret } e' \div A$, as desired.

- **Case** `R-seq-1`**.** Here we have the following derivation:

$$\dfrac{\begin{array}{c}\vdots\\ e \hookrightarrow e'\end{array}}{x \leftarrow e; c \hookrightarrow x \leftarrow e'; c}$$

By hypothesis, we must have a derivation of the form

$$\dfrac{\begin{array}{c}\vdots\\ \vdash e : \Diamond B\end{array} \qquad \begin{array}{c}\vdots\\ \cdot; x{:}B \vdash c \div A\end{array}}{\vdash x \leftarrow e; c \div A}$$

Applying the first part of this theorem, we deduce that $\vdash e' : \Diamond B$, which allows us to deduce that $\vdash x \leftarrow e'; c \div A$.

- **Case** `R-seq-2`**.** Here, our derivation has the following form:

$$\dfrac{\begin{array}{c}\vdots\\ c_1 \hookrightarrow c_1'\end{array}}{(x \leftarrow \mathbf{do}\ c_1); c_2 \hookrightarrow (x \leftarrow \mathbf{do}\ c_1'); c_2}$$

By hypothesis, we must also have a derivation of the form

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash c_1 \div B\end{array}}{\vdash \mathbf{do}\ c_1 : \Diamond B} \qquad \begin{array}{c}\vdots\\ \cdot; x{:}B \vdash c_2 \div A\end{array}}{\vdash (x \leftarrow \mathbf{do}\ c_1); c_2 \div A}$$

Here we can apply the inductive hypothesis we can deduce that $\vdash c_1' \div B$ and so that $\vdash \mathbf{do}\ c_1' : \Diamond B$, which in turn allows us to conclude that $\vdash (x \leftarrow \mathbf{do}\ c_1'; c_2) \div A$, as desired.

- **Case** `R-seq-3`**.** Here our derivation has the form

$$\dfrac{}{x \leftarrow \mathbf{do}\ \mathbf{ret}\ v; c_2 \hookrightarrow \langle\!\langle\, \mathbf{ret}\ v/x \,\rangle\!\rangle c_2}$$

so by hypothesis we must have another derivation of the following form:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash v : B\end{array}}{\vdash \mathbf{ret}\ v \div B}}{\vdash \mathbf{do}\ \mathbf{ret}\ v : \Diamond B} \qquad \begin{array}{c}\vdots\\ \cdot; x{:}B \vdash c_2 \div A\end{array}}{\vdash (x \leftarrow \mathbf{do}\ \mathbf{ret}\ v); c_2 \div A}$$

Here we need another substitution lemma for the substitution operation associated to computations. Since $\vdash \mathbf{ret}\ v \div B$ and $\cdot; x{:}B \vdash c_2 \div A$, we must have that $\vdash \langle\!\langle\, \mathbf{ret}\ v/x \,\rangle\!\rangle c_2 \div A$, which can be easily proved by induction and is the desired conclusion in this case.

□

Theorem 2.3.2 (Progress)**.** The following hold:

(i) If $\cdot \vdash e : A$, then either e is a value, or there exists e' such that $e \hookrightarrow e'$.

(ii) If $\cdot \vdash c \div A$, then either c is a value, or there exists c' such that $c \hookrightarrow c'$.

Proof. (i) By rule induction on the derivation of the judgment $\cdot \vdash e : A$. We present some cases.

- **Case** $\texttt{hyp}^*_u$**.** This case is not possible, as modal variables are not typable in an empty modal context.
- **Case** $\Box\texttt{I}$**.** This case is satisfied since boxed terms are values, as specified by rule `V-box`.
- **Case** $\Box\texttt{E}^u$**.** Here, we have a derivation of the form:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash e_1 : \Box A & u{::}A; \cdot \vdash e_2 : B\end{array}}{\vdash \textbf{let box}\ u = e_1\ \textbf{in}\ e_2 : B}$$

  Applying the inductive hypothesis to the first premise yields two possible cases: either e_1 is a value, or there exists e_1' such that $e_1 \hookrightarrow e_1'$. In the first case, e_1 must be of the form $\textbf{box}\, e$ for some term e, so letting $e' = \textbf{let box}\ u = \textbf{box}\, e\ \textbf{in}\ e_2$ and using rule `R-letbox-2` yields the desired result. In the second case, letting $e' = \textbf{let box}\ u = e_1'\ \textbf{in}\ e_2$ and using rule `R-letbox-1` works.
- **Case** $\Diamond\texttt{I}$**.** This case holds since terms of the form $\textbf{do}\ c$ are values by the rule `V-do`.

(ii) By rule induction on the derivation of the judgment $\cdot \vdash c \div A$.

- **Case** `coerce`**.** Here, we have a derivation of the following form:

$$\dfrac{\begin{array}{c}\vdots \\ \vdash e : A\end{array}}{\vdash \textbf{ret}\ e \div A}$$

  If e is a value, we can use rule `V-ret` to deduce the desired conclusion. Otherwise, we must have $e \hookrightarrow e'$ for some other term e', so we can rule `R-ret` to deduce that $\textbf{ret}\ e \hookrightarrow \textbf{ret}\ e'$.
- **Case** $\Diamond\texttt{E}^x$**.** Here, our derivation has the form

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash e : \Diamond B & x{:}B \vdash c \div A\end{array}}{\vdash x \leftarrow e; c \div A}$$

Applying the first part of the theorem to the first premise yields two cases: either e is a value, or there exists e' such that $e \hookrightarrow e'$. In the latter case, rule `R-seq-1` works. Let us consider the case where e is a value, it must be of the form **do** c, so our derivation has the more detailed form:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash c \div B\end{array}}{\vdash \mathbf{do}\ c : \Diamond B} \qquad \begin{array}{c}\vdots\\ x{:}B \vdash c \div A\end{array}}{\vdash x \leftarrow \mathbf{do}\ c; c \div A}$$

Applying the inductive hypothesis to $\vdash c \div B$, we see that it must be a value or there must be some computation c' such that $c \hookrightarrow c'$. In the first case, the only option is **ret** v for some value v, so we can use rule `R-seq-3` to obtain the desired conclusion. Otherwise, rule `R-seq-2` works.

- **Case** $\Box \mathtt{E}^u_p$**.** This case is analogous to the case of $\Box \mathtt{E}^u$ presented in the first part of the theorem.

□

As with the typechecker, our interpreter can be extended to incorporate the operational semantics rules discussed so far. We defer this discussion to the next chapter, where we examine contextual modalities.

3

Contextualizing Modalities: Exploring Contextual Modal Type Theory

In the previous chapter, we studied intuitionistic modal logic and its computational interpretation via the Curry–Howard correspondence. We saw how the modalities of necessity and possibility allow reasoning not only about what is true, but also about what must or might be true.

We now bring this development to its culmination by introducing Contextual Modal Type Theory (CMTT), in which modalities are relativized to explicit contexts. In this setting, a judgment of the form $[\Psi]A$ `true` asserts that A holds whenever all assumptions in Ψ hold and no others. This generalizes the judgment $\Box A$ `true`, which, in the contextual setting, corresponds to $[\cdot]A$ `true`.

The chapter begins by adapting the natural deduction rules for modal logic to incorporate contextual modalities. We then present a bidirectional formulation of these rules, which enables the extension of our typechecker to handle contextual modal programs. Finally, we extend the operational semantics accordingly and show how to execute programs in this enriched setting.

3.1 Intuitionistic Contextual Modal Logic

Modal logic, with its notions of necessity and possibility, extends propositional logic to reason not only about what is true, but also about what must be true or could be true. In many situations, however, the truth of a proposition depends on the context in which it is formulated. That is, it is not enough to know whether a proposition is true or false in absolute terms; it is necessary to consider the hypotheses under which the proposition is evaluated. As Nanevski, Pfenning, and Pientka (2008) emphasize, the notion of context is fundamental in fields such as linguistics, artificial intelligence, and logic, and it plays a central role when one acknowledges that the truth of propositions depends on the conditions under which they are stated.

The logic given by the rules in Figure 3.1 incorporates the notion of context into intu-

itionistic modal logic. In this setting, contexts are identified with the sets of propositions that hold within them, and they become an explicit part of the logical language.

$$\frac{\Delta, u{::}A\ \mathtt{valid}[\Psi], \Delta'; \Gamma \vdash \Psi}{\Delta, u{::}A\ \mathtt{valid}[\Psi], \Delta'; \Gamma \vdash A\ \mathtt{true}}\ \mathtt{hyp}^*_u$$

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

$$\frac{\Delta; \Gamma \vdash \Psi \qquad \Delta; \Gamma \vdash A\ \mathtt{true}}{\Delta; \Gamma \vdash A\ \mathtt{poss}\langle\Psi\rangle}\ \mathtt{coerce}$$

$$\frac{\Delta; \Gamma \vdash [\Psi]A\ \mathtt{true} \qquad \Delta, u{::}A\ \mathtt{valid}[\Psi]; \Gamma \vdash B\ \mathtt{poss}\langle\Theta\rangle}{\Delta; \Gamma \vdash B\ \mathtt{poss}\langle\Theta\rangle}\ \Box\mathtt{E}^u_p$$

$$\frac{\Delta; \Gamma \vdash A\ \mathtt{poss}\langle\Psi\rangle}{\Delta; \Gamma \vdash \langle\Psi\rangle A\ \mathtt{true}}\ \Diamond\mathtt{I} \qquad \frac{\Delta; \Gamma \vdash \langle\Psi\rangle A\ \mathtt{true} \qquad \Delta; \Psi, x{:}A\ \mathtt{true} \vdash B\ \mathtt{poss}\langle\Theta\rangle}{\Delta; \Gamma \vdash B\ \mathtt{poss}\langle\Theta\rangle}\ \Diamond\mathtt{E}^x$$

$$\frac{\Delta; \Gamma \vdash B_1\ \mathtt{true} \qquad \cdots \qquad \Delta; \Gamma \vdash B_m\ \mathtt{true}}{\Delta; \Gamma \vdash y_1{:}B_1\ \mathtt{true}, \cdots, y_m{:}B_m\ \mathtt{true}}\ \mathtt{ctx}$$

Figure 3.1: Natural deduction rules for contextual modalities

Let us consider a few example derivations.

Example 3.1.1. • $\vdash [A]A\ \mathtt{true}$.

$$\frac{\dfrac{}{x{:}A\ \mathtt{true} \vdash A\ \mathtt{true}}\ \mathtt{hyp}_x}{\vdash [A]A\ \mathtt{true}}\ \Box\mathtt{I}$$

• $[C, C]A \to [C]A\ \mathtt{true}$.

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

- $\vdash [C]A \to [D][C]A$ true.

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

We interpret this logic as a type theory by annotating the natural deduction rules with proof terms, giving rise to Contextual Modal Type Theory. The typing rules are given in Figure 3.2.

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma : \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash u\langle\sigma\rangle : A}\,\texttt{hyp}^*_u$$

$$\frac{\Delta; \Psi \vdash e : A}{\Delta; \Gamma \vdash \textbf{box}\ \Psi.\ e : [\Psi]A}\,\Box\texttt{I} \qquad \frac{\Delta; \Gamma \vdash e_1 : [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash e_2 : B}{\Delta; \Gamma \vdash \textbf{let box}\ u = e_1\ \textbf{in}\ e_2 : B}\,\Box\texttt{E}^u$$

$$\frac{\Delta; \Gamma \vdash \sigma : \Psi \qquad \Delta; \Gamma \vdash e : A}{\Delta; \Gamma \vdash \textbf{ret}\langle\sigma, e\rangle \div A\langle\Psi\rangle}\,\texttt{coerce}$$

$$\frac{\Delta; \Gamma \vdash e : [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash c \div B\langle\Theta\rangle}{\Delta; \Gamma \vdash \textbf{let box}\ u = e\ \textbf{in}\ c \div B\langle\Theta\rangle}\,\Box\texttt{E}^u_p$$

$$\frac{\Delta; \Gamma \vdash c \div A\langle\Psi\rangle}{\Delta; \Gamma \vdash \textbf{do}\ c : \langle\Psi\rangle A}\,\Diamond\texttt{I} \qquad \frac{\Delta; \Gamma \vdash e : \langle\Psi\rangle A \qquad \Delta; \Psi, x{:}A \vdash c \div B\langle\Theta\rangle}{\Delta; \Gamma \vdash \Psi, x \leftarrow e; c \div B\langle\Theta\rangle}\,\Diamond\texttt{E}^x$$

$$\frac{\Delta; \Gamma \vdash e_1 : B_1 \qquad \cdots \qquad \Delta; \Gamma \vdash e_m : B_m}{\Delta; \Gamma \vdash (y_1 \leftarrow e_1, \ldots, y_m \leftarrow e_m) : (y_1{:}B_1, \ldots, y_m{:}B_m)}\,\texttt{ctx}$$

Figure 3.2: Proof term annotations for contextual modalities

3.2 Typechecking Programs

In Figure 3.3, we present the bidirectional formulation of the typing rules presented in the previous section.

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma \Leftarrow \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash u\langle\sigma\rangle \Rightarrow A}\ \texttt{T-mvar}$$

$$\frac{\Delta; \Psi \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{box}\ \Psi.\ e \Leftarrow [\Psi]A}\ \texttt{T-box} \qquad \frac{\Delta; \Gamma \vdash e_1 \Rightarrow [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash e_2 \Leftarrow B}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \Leftarrow B}\ \texttt{T-letbox}$$

$$\frac{\Delta; \Gamma \vdash \sigma \Leftarrow \Psi \qquad \Delta; \Gamma \vdash e \Leftarrow A}{\Delta; \Gamma \vdash \mathbf{ret}\langle\sigma, e\rangle \overset{\bullet}{\Leftarrow} A\langle\Psi\rangle}\ \texttt{T-coerce}$$

$$\frac{\Delta; \Gamma \vdash e \Rightarrow [\Psi]A \qquad \Delta, u{::}A[\Psi]; \Gamma \vdash c \overset{\bullet}{\Leftarrow} B\langle\Theta\rangle}{\Delta; \Gamma \vdash \mathbf{let\ box}\ u = e\ \mathbf{in}\ c \overset{\bullet}{\Leftarrow} B\langle\Theta\rangle}\ \texttt{T-letbox-p}$$

$$\frac{\Delta; \Gamma \vdash c \overset{\bullet}{\Leftarrow} A\langle\Psi\rangle}{\Delta; \Gamma \vdash \mathbf{do}\ c \Leftarrow \langle\Psi\rangle A}\ \texttt{T-do} \qquad \frac{\Delta; \Gamma \vdash e \Rightarrow \langle\Psi\rangle A \qquad \Delta; \Psi, x{:}A \vdash c \overset{\bullet}{\Leftarrow} B\langle\Theta\rangle}{\Delta; \Gamma \vdash \Psi, x \leftarrow e; c \overset{\bullet}{\Leftarrow} B\langle\Theta\rangle}\ \texttt{T-seq}$$

$$\frac{\Delta; \Gamma \vdash e_1 \Leftarrow B_1 \qquad \cdots \qquad \Delta; \Gamma \vdash e_m \Leftarrow B_m}{\Delta; \Gamma \vdash (y_1 \leftarrow e_1, \ldots, y_m \leftarrow e_m) \Leftarrow (y_1{:}B_1, \ldots, y_m{:}B_m)}\ \texttt{T-ctx}$$

Figure 3.3: Bidirectional rules for contextual modalities

To extend our typechecker, we begin by updating the signatures of the `synth` and `check` functions, and by introducing a new function `checkPoss` to handle rules of the form $c \overset{\bullet}{\Leftarrow} A\langle\Psi\rangle$. We also add a `checkSubs` function to implement rules of the form $\sigma \Leftarrow \Psi$. The updated set of signatures is shown in Code Snippet 3.1.

```haskell
type Ctx = [(String, Type)]

type ModCtx = [(String,(Type, Ctx))]

type Subs = [(String, Term)]

synth :: ModCtx -> Ctx -> Term -> Either TypeError Type

check :: ModCtx -> Ctx -> Term -> Type -> Either TypeError ()

checkSubs :: ModCtx -> Ctx -> Subs -> Ctx -> Either Error ()

checkPoss :: ModCtx -> Ctx -> Term -> Type -> Ctx -> Either Error ()
```

Code Snippet 3.1: Typing contexts and typechecking function signatures

The `synth` function remains the main entry point. To typecheck a program, we supply a sufficiently annotated term e and call `synth` with empty contexts as its first two arguments. In other words, we determine whether $\cdot;\cdot \vdash e \Rightarrow A$ holds for some type A. Depending on the bidirectional rules, `synth` may then invoke other functions to complete the typechecking process. Code Snippet 3.2 shows two cases in the definition of `check`.

```haskell
check :: ModCtx -> Ctx -> Term -> Type -> Either TypeError ()
-- T-box
check modCtx ctx (Box psi e) t =
  case t of
    BoxTy psi' ty
    | eqCtx psi psi' -> check modCtx psi' e ty
    | otherwise      -> Left(DifferentContexts psi psi')
    _ -> Left (NotABoxType t)
-- T-letbox
check modCtx ctx (LetBox u e1 e2) t = do
  t1 <- synth modCtx ctx e1
  case t1 of
    BoxTy psi ty -> check ((u, (ty, psi)) : modCtx) ctx e2 t
    _  -> Left (NotABoxType t1)
```

Code Snippet 3.2: Implementation of the rules `T-box`and `T-letbox`

As in the previous chapter, the theorems relating our original typing judgments with

their bidirectional formulations (cf. Theorems 2.2.2 and 2.2.1) continue to hold and can be proved by straightforward induction on derivations.

3.3 Running Programs

This final section presents the missing pieces we need to run contextual modal programs. We begin with additional rules for the judgment e `value`.

$$\frac{}{\mathbf{box}\ \Psi.\ e\ \texttt{value}}\ \texttt{V-box}$$

$$\frac{\sigma\ \texttt{value} \qquad e\ \texttt{value}}{\mathbf{ret}\langle\sigma, e\rangle\ \texttt{value}}\ \texttt{V-ret} \qquad \frac{}{\mathbf{do}\ c\ \texttt{value}}\ \texttt{V-do}$$

$$\frac{e_1\ \texttt{value} \qquad \cdots \qquad e_m\ \texttt{value}}{(y_1 \leftarrow e_1, \ldots, y_m \leftarrow e_m)\ \texttt{value}}\ \texttt{V-subs}$$

Figure 3.4: Additional inference rule for the judgment e `value`

In addition to the values listed above, empty substitutions are also considered values. Our operational semantics rules are similar to those in the previous chapter, but now take into account contexts and substitutions:

$$\frac{\sigma \hookrightarrow \sigma'}{\mathbf{ret}\langle\sigma, e\rangle \hookrightarrow \mathbf{ret}\langle\sigma', e\rangle}\ \texttt{R-ret-1} \qquad \frac{e \hookrightarrow e'}{\mathbf{ret}\langle\sigma, e\rangle \hookrightarrow \mathbf{ret}\langle\sigma, e'\rangle}\ \texttt{R-ret-2}$$

$$\frac{e \hookrightarrow e'}{\Psi, x \leftarrow e; c \hookrightarrow \Psi, x \leftarrow e'; c}\ \texttt{R-seq-1} \qquad \frac{c_1 \hookrightarrow c_1'}{\Psi, x \leftarrow \mathbf{do}\ c_1; c_2 \hookrightarrow \Psi, x \leftarrow \mathbf{do}\ c_1'; c_2}\ \texttt{R-seq-2}$$

$$\frac{}{\Psi, x \leftarrow \mathbf{do}\ \mathbf{ret}\langle v_1, v_2\rangle; c_2 \hookrightarrow \langle\langle \mathbf{ret}\langle\sigma, v\rangle / \langle\Psi, x\rangle\rangle\rangle c_2}\ \texttt{R-seq-3}$$

$$\frac{e_1 \hookrightarrow e_1'}{\mathbf{let\ box}\ u = e_1\ \mathbf{in}\ e_2 \hookrightarrow \mathbf{let\ box}\ u = e_1'\ \mathbf{in}\ e_2}\ \texttt{R-box-1} \qquad \frac{}{\mathbf{let\ box}\ u = \mathbf{box}\ \Psi.\ e\ \mathbf{in}\ e_2 \hookrightarrow [\![\Psi.e/u]\!]e_2}\ \texttt{R-box-2}$$

$$\frac{e_1 \hookrightarrow e_1'}{(y_1 \leftarrow e_1, \sigma) \hookrightarrow (y_1 \leftarrow e_1', \sigma)}\ \texttt{R-subs-1} \qquad \frac{\sigma \hookrightarrow \sigma'}{(y_1 \leftarrow v_1, \sigma) \hookrightarrow (y_1 \leftarrow v_1, \sigma')}\ \texttt{R-subs-2}$$

Figure 3.5: Operational semantics rules for contextual modal constructs

The substitution operations must now take the context Ψ into account. A substitution σ is so named because it performs a simultaneous substitution on a term. Their definitions are given by Nanevski, Pfenning, and Pientka (2008) and are implemented in the source

code accompanying this project. As in the previous chapter, theorems of progress and preservation hold for the contextual calculus and can be proved by induction on derivations. Once the substitutions are in place, implementing the interpreter amounts to writing down the operational semantics rules. Code Snippet 3.3 shows the implementation of the rules `R-box-1` and `R-box-2`, where `modSubstitute` is a function implementing contextual modal substitution.

```
eval :: Term -> Term
-- R-box-1, R-box-2
eval (LetBox u e1 e2) =
  case eval e1 of
    Box ctx e3 -> modSubstitute e2 u ctx e3
    e1' -> LetBox u e1' e2

-- In other cases, the argument must be a value
eval t = t

fullEval :: Term -> Term
fullEval t =
let t' = eval t
in if t == t' then t else fullEval t'
```

Code Snippet 3.3: Implementation of rules `R-box-1` and `R-box-2`

4 Conclusions and Future Work

This thesis has contributed to the study of modal logic and type theory from a computational perspective, addressing both theoretical and practical aspects. First, the modal type system proposed by Davies and Pfenning (2001) was reformulated as a bidirectional type system. This presentation makes the implementation of a typechecker straightforward by separating typing judgments into checking and synthesis modes. Building on this, we presented Contextual Modal Type Theory (CMTT) in a bidirectional style, capturing both modalities of contextual necessity and possibility.

In both the modal and contextual modal systems, we defined operational semantics that specify how well-typed programs can be evaluated. To explore these systems, we implemented both a typechecker and an interpreter in Haskell. The full implementation is available in the public repository accompanying this thesis (López-Aquino, 2025).

These contributions establish a solid foundation for further research in more expressive modal systems. One especially promising direction is the study of Effectful Contextual Modal Type Theory (ECMTT), proposed by Zyuzin and Nanevski (2021). ECMTT extends CMTT by incorporating algebraic effects and handlers, enriching the expressiveness of the type system while maintaining its modal foundation. A natural continuation of this work would be to develop a bidirectional formulation of ECMTT and to discover its logical counterpart via the Curry–Howard correspondence. This would involve identifying a suitable Kripke semantics and a corresponding proof system that reflect its type-theoretic structure.

Bibliography

Belnap, Nuel (1962). "Tonk, Plonk and Plink." In: *Analysis* 22.6, pp. 130–134. DOI: 10.1093/analys/22.6.130.

Davies, Rowan and Frank Pfenning (May 2001). "A modal analysis of staged computation." In: *J. ACM* 48.3, pp. 555–604. ISSN: 0004-5411. DOI: 10.1145/382780.382785. URL: https://doi.org/10.1145/382780.382785.

de Paiva, Valeria (Mar. 2015). *Intuitionistic Modal Logic: 15 Years Later...* Colloquium presentation, University of California, Berkeley. Accessed July 2025. URL: https://logic.berkeley.edu/colloquium/dePaivaSlides.pdf.

Dummett, Michael (1991). *The Logical Basis of Metaphysics*. Cambridge: Harvard University Press.

Dunfield, Jana and Neel Krishnaswami (May 2021). "Bidirectional Typing." In: *ACM Computing Surveys* 54.5, pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3450952. URL: http://dx.doi.org/10.1145/3450952.

Francez, Nissim and Roy Dyckhoff (2012). "A Note on Harmony." In: *Journal of Philosophical Logic* 41.3, pp. 613–628. DOI: 10.1007/s10992-011-9208-0.

Gentzen, Gerhard (1935). "Untersuchungen über das logische Schließen. I." In: *Mathematische Zeitschrift* 39.1, pp. 176–210. ISSN: 1432-1823. DOI: 10.1007/BF01201353. URL: https://doi.org/10.1007/BF01201353.

López-Aquino, Kevin (2025). *cmtt*. https://github.com/kevinlopaq/cmtt. Accessed: 2025-06-27.

Martin-Löf, Per (1987). "Truth of a Proposition, Evidence of a Judgement, Validity of a Proof." In: *Synthese* 73.3, pp. 407–420. DOI: 10.1007/bf00484985.

Martin-Löf, Per (1994). "Analytic and Synthetic Judgements in Type Theory." In: *Kant and Contemporary Epistemology*. Ed. by Paolo Parrini. Dordrecht: Springer Netherlands, pp. 87–99. ISBN: 978-94-011-0834-8. DOI: 10.1007/978-94-011-0834-8_5. URL: https://doi.org/10.1007/978-94-011-0834-8_5.

— (1996). "On the Meanings of the Logical Constants and the Justifications of the Logical Laws." In: *Nordic Journal of Philosophical Logic* 1.1, pp. 11–60.

Nanevski, Aleksandar, Frank Pfenning, and Brigitte Pientka (June 2008). "Contextual modal type theory." In: *ACM Trans. Comput. Logic* 9.3. ISSN: 1529-3785. DOI: 10.1145/1352582.1352591. URL: https://doi.org/10.1145/1352582.1352591.

Pfenning, Frank (2009). *Lecture 3: Harmony*. Accessed: 2025-05-10. URL: https://www.cs.cmu.edu/~fp/courses/15317-f09/lectures/03-harmony.pdf.

Pfenning, Frank and Rowan Davies (Aug. 2001). "A judgmental reconstruction of modal logic." In: *Mathematical. Structures in Comp. Sci.* 11.4, pp. 511–540. ISSN: 0960-1295. DOI: 10.1017/S0960129501003322. URL: https://doi.org/10.1017/S0960129501003322.

Pierce, Benjamin C. and David N. Turner (1997). *Local Type Inference*. Tech. rep. CSCI No. 493. Indiana University.

— (1998). "Local Type Inference." In: *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM Press, pp. 252–265.

— (Jan. 2000). "Local Type Inference." In: *ACM Trans. Program. Lang. Syst.* 22.1, pp. 1–44. ISSN: 0164-0925. DOI: 10.1145/345099.345100. URL: https://doi.org/10.1145/345099.345100.

Prawitz, Dag (1965). *Natural Deduction: A Proof-Theoretical Study*. Mineola, N.Y.: Dover Publications.

Prior, A. N. (1960). "The Runabout Inference-Ticket." In: *Analysis* 21.2, pp. 38–39. DOI: 10.1093/analys/21.2.38.

Russell, Bertrand (1905). "Necessity and Possibility." In: *Toward the "Principles of Mathematics", 1900–1902*. Ed. by Gregory H. Moore. Vol. 3. Collected Papers of Bertrand Russell (1993). Originally delivered as a lecture in 1905. London: Routledge, pp. 508–520.

Scott, Dana (1970). "Advice on Modal Logic." In: *Philosophical Problems in Logic: Some Recent Developments*. Ed. by Karel Lambert. Dordrecht: Springer Netherlands, pp. 143–173. ISBN: 978-94-010-3272-8. DOI: 10.1007/978-94-010-3272-8_7. URL: https://doi.org/10.1007/978-94-010-3272-8_7.

Xie, Ningning and Bruno C. d. S. Oliveira (2018). "Let Arguments Go First." In: *Programming Languages and Systems*. Ed. by Amal Ahmed. Cham: Springer International Publishing, pp. 272–299. ISBN: 978-3-319-89884-1.

Zyuzin, Nikita and Aleksandar Nanevski (Aug. 2021). "Contextual modal types for algebraic effects and handlers." In: *Proc. ACM Program. Lang.* 5.ICFP. DOI: 10.1145/3473580. URL: https://doi.org/10.1145/3473580.

Term Index

Notation Index